Fachhochschule Stuttgart

Hochschule der Medien

Diplomarbeit im Studiengang Medieninformatik

Entwicklung eines webbasierten Buchungssystems (Digitales Logbuch) für den Akademischen Segler-Verein Hamburg e.V.

Vorgelegt bei	Prof. Dr. Walter Kriha (1. Prüfer) Dr. Tim Eberhardt (2. Prüfer)
von	Steffen Hartmann

Stuttgart, Februar 2005

Erklärung

Hiermit erkläre und versichere ich, dass ich diese Diplomarbeit selbstständig und ohne fremde Hilfe verfasst habe. Es wurden keine anderen Hilfsmittel als die aufgeführten benutzt. Sämtliche verwendeten Quellen sind im Text oder im Anhang als solche ausgewiesen.
Darüber hinaus bestätige ich, dass diese Arbeit bisher weder im In- noch im Ausland als Prüfungsarbeit eingereicht wurde.

_____ _____

Datum Unterschrift

Abstract

Dieses vorliegende Dokument stellt den schriftlichen Teil der Diplomarbeit dar, gleichzeitig ist es aber auch als Dokumentation zu betrachten. Es richtet sich hauptsächlich an Systemarchitekten und Software-Entwickler, um einen Überblick darüber zu geben, wie die entwickelte Buchungs-Software aufgebaut ist. Dieses Dokument dient als Grundlage und Referenz; es sollte vor dem Einstieg in den Source Code gelesen und verstanden werden, denn es erläutert die grundlegenden Abläufe, Modelle und Praktiken.

Daneben existieren weitere Dokumente, die tieferen Einblick geben oder spezielle technische Probleme anhand von Verfahrensanweisungen lösen helfen. Diese Dokumente befinden sich (neben Source Code, generiertem JavaDoc und den in dieser Diplomarbeit verwendeten Diagrammen und Bildern) auf der beiliegenden CD-ROM.

Wie bereits angedeutet, soll diese schriftliche Ausarbeitung den Leser befähigen, sich in die umfangreiche Software einzuarbeiten (z.B. für Wartungsarbeiten oder Weiterentwicklung der Software). Deshalb werden grundlegende Kenntnisse und Begriffe der Programmiersprache Java, von Entwurfsmustern (Design Patterns), von Betriebssystemen (Linux und Windows) und Internet-Technologien (Web- und Applicationserver, Protokolle, HTML, CSS, etc.) vorausgesetzt. Die Kapitel 2 bis 4 können darüber nur einen groben Überblick geben und gehen nur selten richtig ins Detail. Für Interessierte, die sich eingehender damit beschäftigen wollen, existiert eine Vielzahl an weiterführender Literatur.

Kapitel 5 geht ausführlich auf die Anforderungen der Software ein, so wie sie in einem Pflichtenheft hätten definiert worden sein können. Besonders hier kommen viele großformatige UML-Diagramme (Unified Modeling Language) zum Einsatz, sind sie doch aussagekräftig und erschließen sich schnell wie einfach dem Betrachter.

In Kapitel 6 wird dann erläutert, wie die Geschäftsprozesse aus dem vorhergehenden Kapitel umgesetzt wurden. Es handelt sich dabei hauptsächlich um Komponenten, die auf dem Server installiert sind und den wesentlichen Teil der Software bilden (Teile davon werden allerdings von anderen, lokalen Komponenten wiederverwendet).

Eine solche lokale Komponente ist das Import-Export-Tool (Kapitel 7) – eine reine Java-Applikation mit einfacher Benutzeroberfläche. Sie ermöglicht den regelmäßigen Austausch von Mitgliederdaten zwischen der Online-Datenbank MySQL und der Offline-Datenbank Access. Dies ist notwendig, da der Segelverein, für den diese Software erstellt wurde, auf die bisher verwendete Access-Datenbank nicht verzichten konnte. Deshalb kann oder muss man hier durchaus von Individual-Software sprechen. Allerdings ist es auch möglich, auf die Offline-Datenbank zu verzichten, was den Einsatz der Buchungs-Software bei anderen Segelvereinen mit ähnlichen Geschäftsprozessen ermöglicht.

Kapitel 8 widmet sich einer Zusatz-Software (ebenfalls eine Java-Applikation), während Kapitel 9 eine Schlussbetrachtung und persönliche Erfahrungen des Autors enthält.

Danksagung

An dieser Stelle möchte ich denen herzlich danken, die zum Entstehen der Software und der Diplomarbeit direkt oder indirekt beigetragen haben. Das sind

- Herr Prof. Dr. Walter Kriha von der Hochschule der Medien, der mit seiner freundlichen und enthusiastischen Art dieses Projekt von Anfang an begleitet hat.
- Herr Dr. Tim Eberhardt als Geschäftsführer der Firma MAGTIME synergy associates in Hamburg, der Ansprech- und Projektpartner war. Ohne ihn wäre dieses Projekt nicht zu Stande gekommen.
- Frau Maria Ganescu, ohne deren Vermittlung die Diplomarbeit ebenso wenig zustande gekommen wäre.
- die Herren Dipl.-Ing. (FH) Michael Täge, Dipl.-Ing. (FH) Uwe Laufs, Dipl.-Ing. (FH) Tilman Hansen und Alexander Schwarz, die mit ihren Tipps und Denkanstößen dazu beitrugen, dass das Projekt bei design- und programmiertechnischen Problemen nicht für längere Zeit ins Stocken kam.
- Herr Nicolas Schmid, Student an der HdM, der die Lösung zu einem Datenbank-Problem wusste, an dem ich über mehrere Tage hinweg fast verzweifelt wäre.
- Frau Maraike Finnern, die freundlicherweise das Korrekturlesen der schriftlichen Ausarbeitung übernahm.
- Familie Plätschke, die mir kurzfristig Unterkunft in der Nähe von Hamburg anbot.
- meine Mutter und mein Großvater, die mich während der Diplomarbeit finanziell unterstützten.

Hinweise zum Lesen dieses Dokuments

- Verzeichnisse, Pfade, Dateinamen, Klassennamen, Variablennamen, Anweisungen, Benutzereingaben und Code-Zeilen werden zur besseren Unterscheidung in `nicht-proportionaler` Schrift dargestellt. Mehrere Code-Zeilen und Programm-Listings sind fast immer um Kommentare und oft auch um „uninteressante“ Code-Zeilen gekürzt und werden mit `kleiner nicht-proportionaler` Schrift dargestellt.
- Parameter von Java-Methoden sind nicht unbedingt aufgeführt.
- Auf der beiliegenden CD befindet sich neben einem PDF dieses Drucks sämtlicher Source Code[1], die JavaDoc-Dokumentation und einige PDFs, die als „Kochrezept“ zu betrachten und zu benutzen sind. In ihnen wurden die Erkenntnisse und konkreten Vorgänge z.B. bei der Linux-Administration gesammelt.
- Alle hier abgedruckten Bilder und Diagramme und weitere, ungedruckte Diagramme sind ebenfalls auf der CD zu finden, da sie oft wegen ihrer Größe in gedruckter Form nicht so gut zu erkennen sind wie als Bilddatei am Bildschirm.

[1] Nicht beim „öffentlichen“ Exemplar für die Bibliothek.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Die Idee zur umgesetzten Software schlummerte (nach eigener Aussage) schon länger in der Schreibtischschublade von Herrn Dr. Eberhardt, seines Zeichens Geschäftsführer der Firma MAGTIME synergy associates und Vorstandsmitglied des Akademischen Segler-Vereins Hamburg. Verstand ich unter dieser Idee zu Beginn des Projekts noch eher eine elektronische Liegeplatzverwaltung, so wurde daraus im Laufe der Zeit das Digitale Logbuch, kurz DiLog.
Dabei war das Digitale Logbuch nicht auf eine bestimmte Programmiersprache festgelegt: Es war lediglich klar, dass es sich um ein Client-Server-Modell handeln musste, bei dem als Client ein gängiger Webbrowser zum Einsatz kommt, denn die Verteilung einer Client-Software an ca. 500 Vereinsmitglieder wäre schlichtweg unpraktikabel.

Zudem muss bedacht werden, dass jedes Software-Upgrade (infolge eines Versionssprunges) oder Software-Update (Bugfixes und kleinere technische Änderungen) nochmals eine anschließende Verteilung des geänderten Codes nach sich zieht. Selbst wenn man – um sich die kostenträchtige Auslieferung auf einem Wechseldatenträger wie CD-ROM zu sparen – eine Möglichkeit zum direkten Download anbieten würde, müsste der Benutzer den Patch doch selbst einspielen oder die Installation anstoßen oder zumindest das automatische Aufspielen der Änderungen genehmigen[2]. Welches der eben geschilderten Modelle man auch favorisiert, ein Problem bleibt immer: Das System in seiner Gesamtheit wird zunehmend heterogen, denn es ist nicht davon auszugehen, dass jeder Benutzer ein Update sofort oder überhaupt mitmacht. Es bliebe die letzte Möglichkeit, den Benutzer durch „Nutzungsverbot“ zu zwingen, jedoch erzeugt das nicht selten großen Unmut. Besonders in der Einführungsphase – wo naturgemäß die meisten Fehler zu erwarten sind – geht es darum, die Akzeptanz des Nutzers zu erlangen; somit scheidet auch diese Möglichkeit aus!

Außerdem gibt es in einem Segelverein besseres zu tun, als die Mitglieder im Umgang mit einer neuen Software zu schulen. Man muss davon ausgehen, dass nicht jeder im Umgang mit Computern geübt ist. Da man davon ausgehen kann, dass die Meisten bereits mit dem Internet mehr oder weniger intensiven Kontakt hatten, zwingt es sich (neben den bereits genannten Gründen) förmlich auf, einen der aktuellen Webbrowser[3] als Ausgangspunkt zu definieren. Natürlich nimmt die Vielfalt von sog. Look & Feel der Websites im World Wide Web mit fortschreitenden technischen Möglichkeiten[4] ständig zu, die grundsätzliche Bedienung ist jedoch fast immer dieselbe. Es wurde deshalb auch bewusst ein Webdesign gewählt, das eher schlicht daherkommt, leicht zu durchschauen und intuitiv zu bedienen ist.

---

2 Obwohl es komplett benutzerunfreundlich ist, verfolgen selbst große Firmen teilweise den Ansatz, Software-Updates ohne vorhergehende Benachrichtigung des Benutzers automatisch zu installieren. Dies kann schnell zu Verunsicherungen gegenüber der Firma und der Software führen.

3 Die Software wurde mit folgenden Browsern getestet: Microsoft Internet Explorer 6.0, Netscape 6.2, Mozilla 1.7, Opera 7.54

4 JavaScript, Cascading Style Sheets (CSS), Layer-Techniken, Java-Applets, Multimedia-Inhalte usw.

1.1 Auswahl der Programmiersprache

Nachdem die Client-Server-Architektur mit webbasiertem Ansatz als einzige praktikable Lösung feststand, stellte sich die Frage der Programmiersprache. In den Kreis der engeren Kandidaten kamen Perl, PHP, Active Server Pages (ASP) und Java (Servlets und Java Server Pages). [Hall 2001] (S. 29-32) beschreibt die Vorzüge der Java-Technologie gegenüber PHP, ASP und ColdFusion,[5] was hier kurz zusammengefasst und ergänzt werden soll:

Vorteile von Servlets gegenüber „traditionellem“ CGI (Common Gateway Interface):

- Effizienz: Bei einer HTTP-Anfrage (Hypertext Transfer Protocol) wird nur ein leichtgewichtiger Java-Thread und kein schwergewichtiger Prozess des Betriebssystems gestartet.
- Persistenz: Nach Ablauf des Programms endet CGI im Gegensatz zu einem Servlet, was die Datenhaltung und Connection-Pooling[6] erschwert.
- Zweckmäßigkeit: Servlets bringen viele bordeigene Mittel mit für das Session-Management, oder um HTML-Formulardaten (Hypertext Markup Language) automatisch zu parsen, Header-Daten zu lesen/schreiben und um Cookies zu behandeln (siehe Kapitel 4.1).
- Mächtigkeit: Servlets können mit dem Webserver direkt kommunizieren, was CGI-Programme nicht oder nur mit auf den speziellen Server zugeschnittener Zusatz-Software können.
- Portierbarkeit: Servlets basieren auf gewissen Standard-APIs (Application Programming Interface). Ein Servlet läuft ohne Anpassungen auf jedem Webserver, der den betreffenden Standard bietet (und das sind nicht wenige Server).
- Sicherheit: Auf dem Server werden CGI-Programme oft von Sprachen verarbeitet, die Array- oder Zeichenkettengrenzen nicht prüfen (z.B. C oder C++). Daher können Angreifer direkt im Programmspeicher schreiben und gefährliche Speicherüberläufe herbeiführen. Das ist bei Java nicht der Fall.
- Erschwinglichkeit: Es existieren performante und mächtige, aber zugleich kostenlose Server für die Ausführung von Servlets.

Vorteile von Java Server Pages:

- gegenüber ASP (und ColdFusion): Active Server Pages sind auf Microsofts Internet Information Server (IIS) und die Windows-Plattform festgelegt und deshalb nur wenig portabel. VBScript oder ähnliche ASP-spezifische Sprachen sind nicht so mächtig und weniger gut wiederverwendbar als Java. Zudem spricht der Kostenfaktor im direkten Vergleich zu kostenlosen Open-Source-Produkten gegen Microsoft-Produkte.
- gegenüber PHP: Ursprünglich eine Skriptsprache, nunmehr auch objektbasiert, ist für größere Projekte weniger gut geeignet.

[5] Die Beschreibung ist nicht ganz objektiv, aber trotzdem recht informativ.

[6] Eine Verbindung bleibt nach Datenbankabfrage geöffnet und wird mehrfach genutzt, was einen Geschwindigkeitsvorteil bringt.

- gegenüber Perl: Perl ist eine Skriptsprache, die ebenfalls nur für kleinere Projekte sinnvoll zu verwenden ist, schon allein durch die teilweise recht kryptische Syntax.

Nachteile der Java-Technologie:

- Die Einrichtung und Administration eines Servlet-tauglichen Servers ist nicht gerade einfach und kann leicht zu unvorhergesehenen Problemen führen.
- Die Technologie und der Server an sich sind zwar kostenlos, jedoch ist es nicht einfach, einen zuverlässigen und günstigen Webhoster zu finden (was wohl mit vorangehendem Punkt zu begründen ist). Während bald jeder Webhoster CGI und PHP selbst in billigsten Paketen anbietet, sind (günstige) Java-Angebote noch eher rar gestreut. Obwohl zunehmend Bewegung zu erkennen ist, muss man doch meist auf einen teuren Root-Server[7] zurückgreifen, was wiederum nachteilig bewertet werden kann, da man dann für die Sicherheit des Servers, für Backups von Daten usw. selbst verantwortlich ist.

Abschließend muss noch erwähnt werden, dass die hier erwähnten Argumente gegen CGI und die anderen Techniken aus dem Jahr 2000 stammen und schon deshalb nicht mehr ganz aktuell sind. Manche Begründung ist aber auch deshalb nicht mehr haltbar, weil sich CGI inzwischen weiterentwickelt hat (Stichwort FastCGI[8]). Java gewinnt den Vergleich jedoch trotzdem, da auch Java ständig weiterentwickelt wurde und nun zusätzliche Software speziell für Webanwendungen verfügbar ist (z.B. das Framework Struts).

1.2 Aufgabenstellung

Beschreibung des Ist-Zustandes

Der Segelverein hat zur Zeit ca. zehn Binnenschiffe an der Außenalster in Hamburg liegen. Die Vereinsmitglieder können diese Boote jeweils für eine gewisse Zeit mieten. Dazu tragen sie sich in ein Buch ein, das am Bootsanleger ausliegt. Nun kann es bei schönem Wetter gut passieren, dass viel mehr Segler aufkreuzen als Bootskapazität vorhanden ist und viele unverrichteter Dinge abziehen müssen. Deshalb ist es möglich, sich vorauslaufend einzutragen und mit einer Buchung ein Boot zu reservieren. Das erspart dem einzelnen Vereinsmitglied aber nicht den Weg an die Alster, denn Reservierungen müssen vor Ort in das Logbuch eingetragen werden. Ebenso ist man vor Enttäuschungen nicht gefeit, wenn man sich auf gut Glück nach Feierabend oder womöglich an einem freien Tag extra zum Segeln an die Alster aufmacht und feststellen muss, dass alle Boote für den restlichen Tag ausgebucht sind.

Zum „Fuhrpark“ des Vereins gehören derzeit auch vier seegängige Schiffe, die im Yachthafen Wedel beheimatet sind. Für sie gelten andere Regeln. Zu Beginn eines Jahres wird ein Törnplan zusammengestellt, der jede einzelne Segelyacht möglichst gut auslastet

[7] Eine günstige Alternative für kleinere Projekte bieten vereinzelt zu findende sog. virtuelle Root-Server. Dabei teilen sich eine bestimmte Anzahl an Benutzer die Rechen- und Festplattenkapazität eines realen Servers. Allerdings ist die Leistung natürlich dementsprechend niedriger.

[8] http://www.fastcgi.com

und spätestens am Ende der Saison wieder auf die Elbe bringt. Hierbei wird also längerfristig geplant und meist wissen Skipper und Crew schon frühzeitig über die vereinbarten Termine Bescheid, denn schließlich kann solch ein Törn mehrere Tage bis Wochen dauern.

Wenn im Herbst dann das Absegeln stattgefunden hat und damit das Saison-Ende unausweichlich ist, darf der Obmann das („analoge“) Logbuch auswerten: Dazu müssen mühsam alle Einträge sortiert und den einzelnen Vereinsmitgliedern zugeschrieben werden.

Beschreibung des Soll-Zustandes

Das Vereinsmitglied loggt sich zu Hause oder am Arbeitsplatz mit einem Webbrowser über das Internet auf den DiLog-Server ein. Dann gibt der Benutzer seinen Wunschtermin und das gewünschte Boot ein und erfährt in Sekundenschnelle bequem ohne vor Ort sein zu müssen, ob das gewünschte Boot bereits reserviert ist oder noch zur Verfügung steht. Nun gibt der Benutzer seinen Reservierungswunsch in Form einer Buchung ein und bekommt sofort eine Buchungsbestätigung per E-Mail. Andere Benutzer können diese Buchung nun ihrerseits sehen und wissen, dass das Boot für einen bestimmten Zeitraum nicht zur Verfügung stehen wird. Dies gilt für Binnenschiffe an der Außenalster.

Seeschiffe können nicht bis kurz vor Törnbeginn gebucht werden, da die Planung wie schon erwähnt recht langfristig stattfindet. Deshalb wird eine Buchung in diesem Fall nur als Reservierungswunsch betrachtet und ist erst gültig, wenn der für das Boot zuständige Obmann die Buchung bestätigt.

Die Auswertung und Zuordnung der einzelnen Buchungen zu den jeweiligen Vereinsmitgliedern oder Booten dauert nur wenige Sekunden. Dabei kann zwischen einer übersichtlichen HTML-Darstellung und einer druckfreundlichen PDF-Version (Portable Document Format) gewählt werden. Zusätzlich kann die Sortierung beeinflusst werden (Buchungen in Boote gruppiert oder nach Zeit des Buchungsbeginns geordnet).

Voraussetzungen

Dies alles setzt voraus, dass das System (DiLog) über eine Benutzerverwaltung verfügt und somit einzelne Benutzer unterscheiden kann. Das beginnt beim Login des Benutzers, geht über die Vorhaltung von Mitglieds- und Kontaktdaten und endet bei der Vergabe von Segelberechtigungen (die Erlaubnis, ein bestimmtes Schiff führen zu dürfen). Diese Daten müssen alle vom Administrator und – wo immer möglich – zur Arbeitserleichterung auch durch das Vereinsmitglied änderbar sein.

Diese Mitgliederdaten befinden sich in einer Access-Datenbank, und zwar in einer einzigen unnormalisierten Tabelle. Es muss also eine Möglichkeit geschaffen werden, diese Daten einmalig aus dieser Relation zu lesen und in die ausreichend normalisierte MySQL-Datenbank zu schreiben.

Doch damit nicht genug: Die Access-Datenbank enthält auch einige Abfragen (beispielsweise Skipper eines Bootes oder Zugehörigkeit zu Vereinsgruppen ausgeben),

Masken und Formulare zur Datenpflege und Berichte zur Generierung und Druck des Mitgliederverzeichnisses. Diese sind auch weiterhin unverzichtbar. Zudem geht die Datenpflege lokal ohne Online-Verbindung z.T. schneller (und kostengünstiger) von der Hand als die direkte Dateneingabe über HTML-Formulare.
Es muss also ein Weg gefunden werden, wie die beiden Datenbanken synchron gehalten werden können[9], wobei Daten online oder offline eingepflegt und geändert werden können müssen. Zudem muss berücksichtigt werden, dass – will man DiLog auch an andere Segelvereine verkaufen – das System auch ohne die Access-Datenbank arbeiten können muss.

Umsetzung

Gewünscht ist ein lauffähiger Prototyp, der die Funktionalität einer ausgewachsenen Software andeutet, um somit die „konservativen Kräfte“ im Vereinsvorstand von den Vorzügen eines digitalen Logbuchs zu überzeugen.

Tatsächlich entstanden ist ein Endprodukt, das weit – sehr weit – über den Status eines Prototyps hinausgeht! Im Prinzip könnte die entstandene Software mittelfristig in Betrieb gehen, es wurden bisher allerdings keine Langzeit- und Lasttests durchgeführt. Wie das bei Software, die unter Zeitdruck und Budget-Knappheit entwickelt wird, so üblich ist, fehlte eine ausreichende Testphase unter Realbedingungen. Daher sind kleinere Fehler und Unregelmäßigkeiten nicht auszuschließen obwohl das System prinzipiell stabil läuft.

Abb. 1: DiLog-Logo

[9] siehe Kapitel 7 (Import-Export-Tool)

2 Eingesetzte Software

In diesem zweiten Kapitel soll auf die zum Einsatz gekommenen Server, Betriebssysteme, Datenbanken, Verschlüsselung und Programmiersprache eingegangen werden. Dabei werden die einzelnen Themen eher kurz beleuchtet, sie sind lange nicht erschöpfend abgehandelt.

2.1 Open Source Software

Die Open Source Initiative (OSI) ist eine nicht-gewinnorientierte Körperschaft, die sich der Förderung von Open Source Software (OSS) verschrieben hat. Die Idee, die hinter OSS steht, beschreibt [OSI 2005] so:

> *When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.*

Es gibt drei charakteristische Hauptmerkmale für OSS, so beschrieben in der Open Source Definition (OSD)[10]:

- Der Quelltext (auch Source Code) der Software liegt unkompiliert[11] vor. Er ist also von Menschen lesbar (und somit nachvollziehbar).
- Die Software darf beliebig kopiert, verbreitet und genutzt werden. Dabei ist es egal, wie viele Benutzer das Programm benutzen, wie viele verschiedene Installationen existieren, wer es benutzt oder in welchem Umfeld es eingesetzt wird.
- Der Quelltext darf verändert und weitergegeben werden. Dies ist sogar gewollt, denn davon lebt OSS letztendlich.

Dabei bedeutet „open source“ nicht unbedingt kostenlos, obwohl der Großteil an OSS tatsächlich kostenlos erhältlich ist. Weitere Einzelheiten sind in einer Fülle spezieller Lizenzen geregelt. Zu den „klassischen“ Lizenzen gehören die GNU General Public License (GPL), die GNU Lesser General Public License (LGPL) und die BSD-Lizenz. Seit dem Erscheinen von Mozilla 1998 ist die Mozilla Public License sehr verbreitet. Auf die z.T. komplizierten rechtlichen Unterschiede dieser Lizenzen soll hier nicht eingegangen werden.

Kurz zur Geschichte von OSS: 1997 veröffentlichte Eric Raymond (ein Mitbegründer der OSI) seinen Essay „The Cathedral and the Bazaar“. Darin beschreibt [Raymond 1997] die Vor- und Nachteile von Open Source Software anhand der Entstehung des Betriebssystems Linux („Die Linux Gemeinde gleicht schon eher einem grossen plappernden Basar …“). Beeinflusst von diesem Aufsatz entschied Netscape 1998, den Quelltext seines Browsers Netscape Navigator offenzulegen. Hintergrund war die zunehmende Dominanz des

[10] http://www.opensource.org/docs/definition.php

[11] Ein Compiler erstellt aus dem Quell-Code einen binären für Rechner les- und ausführbaren Maschinen-Code.

Browsers aus dem Hause Microsoft. Aus dem veröffentlichten Source Code entstand später der schon erwähnte Browser Mozilla.
Kurz darauf gründeten Raymond, der Computerwissenschaftler B. Perens und T. O'Reilly (Gründer des bekannten gleichnamigen Verlags) die schon vorgestellte Open Source Initiative, um Open Source Software besser zu vermarkten und ihr auch in der Wirtschaft zu Akzeptanz zu verhelfen.

Ein gerne genannter Vorteil von OSS ist der Sicherheitsaspekt; da bei OSS viele Entwickler Einsicht in den Quelltext erhalten, werden eventuelle Fehler und Sicherheitslücken viel schneller entdeckt und beseitigt. Genau andersherum argumentieren große Software-Konzerne[12]: Werden Sicherheitslöcher gefunden und publiziert (wie das bei OSS der Fall ist), so werden sie auch viel eher durch bösartige Angreifer ausgenutzt.

2.2 Programmiersprache Java

Geschichte

Seinen Ursprung hatte Java 1991 in einem „Green" genannten Projekt der amerikanischen Firma Sun Microsystems, das eine vollständige Betriebssystemumgebung zum Ziel hatte. Maßgeblich daran beteiligt waren J. Gosling, M. Sheridan und P. Naughton. Nach einigen Weiterentwicklungen (es entstand eine ziemlich intelligente und leistungsstarke Fernbedienung, die aber keiner produzieren wollte und eine Box für „Video on Demand", wofür die Zeit aber noch nicht reif war) folgte die Ausrichtung auf das World Wide Web als Teil des wachsenden Internets. Es entstand der Browser WebRunner (später HotJava), der komplett in Java geschrieben war. Dabei war er aber auch fähig, Java-Bytecode zu interpretieren (heutzutage spricht man von Applets).

Ein großer Durchbruch kam 1996, als Netscape seinen Webbrowser in Version 2 mit integriertem Java herausbrachte. Wenige Monate später lieferte Sun Java in der Version 1.02 aus. Mit einer eigenständigen Java-Version konnte man nun auch selbstständige Applikationen erstellen und war nicht nur auf im Browser laufende Applets festgelegt. Jedoch war Java noch stark verbesserungsfähig, man konnte beispielsweise laut [Horstmann 1999] (S. 31) nicht einmal drucken.

Mit Bekanntgabe der Version 1.1 auf der JavaOne-Konferenz 1996 in San Francisco, der Version 1.2 (auch Java 2 genannt) zwei Jahre später bei derselben Veranstaltung und den darauf folgenden Versionen bis 1.5 (auch als 5.0 bezeichnet) hat sich daran einiges geändert. Java hat sich angeschickt, eine der bekanntesten Programmiersprachen zu werden und der Boom hält weiter an. Die vorliegende Software DiLog wurde mit Version 1.4.2_04-b05 entwickelt und getestet.

[12] Allen voran versucht Microsoft vehement gegen Open-Source-Entwicklung vorzugehen.

Grundkonzepte von Java

Sun Microsystems hat Java in einem „White Paper“ als „simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language“ bezeichnet ([Java 2005]). Diese Schlagworte sollen hier nun näher beleuchtet werden.

- ***Einfach***
  Heute wird meist – wenn nicht mit Java – in C oder C++ programmiert. Java ist schon rein vom Syntax her an C++ angelehnt. Jedoch ist Java um Header-Dateien, Zeigerarithmetik und –syntax, Strukturen, Unions, Operatoren-Überladung und virtuelle Basisklassen „bereinigt“. Eine große Erleichterung und Hilfe zur Fehlervermeidung stellt der „Garbage Collector“ dar. Er kümmert sich um das Speicher-Management, indem er den Arbeitsspeicher verwaltet, Speicher zuweist und nicht mehr benötigte Objekte automatisch aus dem Speicher entfernt.
  Einfach bedeutet auch klein. Da Java ursprünglich für kleine Geräte gedacht war (man denke an die Fernbedienung) und Speicher früher sehr teuer war, sollen der Java-Interpreter und die minimal erforderliche Klassenunterstützung ca. 40 Kilobyte (KB) beanspruchen. Thread-Unterstützung und grundlegende Standard-Bibliotheken schlagen mit weiteren 175 KB zu Buche [Horstmann 1999] (S. 21).

- ***Objektorientiert***
  Objektorientierung hat sich seit vielen Jahren in nicht wenigen Programmiersprachen bewährt. Da dieses Konzept hilfreich ist, große Software-Projekte zu verwalten, Qualität zu erhöhen, Fehlerraten zu drücken und hohe Wiederverwendbarkeit von Software-Modulen zu erreichen, ist es undenkbar, dass eine moderne Sprache darauf verzichtet. Die Grundidee dabei ist, zu modellierende Dinge der realen Welt softwaretechnisch mit derselben Sicht umzusetzen.

- ***Verteilt***
  Da Java im Internet-Umfeld groß wurde, bietet es einfache Unterstützung für viele Protokolle, die sich auf TCP/IP (Transmission Control Protocol/Internet Protocol) stützen, z.B. HTTP oder FTP (File Transfer Protocol). So lässt sich beispielsweise relativ einfach eine Socket-Verbindung realisieren.

- ***Robust***
  Java erkennt als stark typisierte Sprache schon frühzeitig Typprobleme beim Kompilieren. Zudem werden Laufzeit-Prüfungen durchgeführt. Zeigerfehler, Speicherzuweisungsfehler und Speicherlücken wie bei C++ sind ausgeschlossen.

- ***Sicher***
  Java wurde schon zu Beginn für verteilte Systeme entwickelt, und die eingebauten Sicherheitsmechanismen lassen sich nur sehr schwer umgehen. Mit ein Grund dafür ist sicherlich die Entscheidung, auf Pointer zu verzichten und Array- oder Zeichenkettengrenzen zu überprüfen (siehe Kapitel 1.1, Vorteile von Servlets, Aufzählungspunkt Sicherheit). So kann auch das Lesen und Schreiben lokaler

Dateien verboten werden. Obwohl es sich nicht nachweisen lässt, wird Java als die wohl sicherste existierende Programmiersprache gehandelt (behauptet [Horstmann 1999] (S. 24)).

- ***Architekturneutral***
  Als „Netzwerksprache“ entstammt Java einer naturgemäß heterogenen Rechner-Landschaft. Um auf möglichst allen Rechnerarten und Betriebssystemen zu funktionieren, erstellt der Compiler aus dem Quelltext eines Programms einen sog. Bytecode. Dieser ist architekturneutral und lässt sich plattformunabhängig durch die JVM (Java Virtual Machine) auf jedem Zielsystem ausführen, für das ein JRE (Java Runtime Environment) verfügbar ist. Es gibt für alle gängigen Betriebssysteme wie Linux, Windows , MacOS, Palm-OS und Solaris eine passende JVM. Bytecode geht zwar zu Lasten der Geschwindigkeit, jedoch gleichen moderne JIT-Compiler (Just In Time) das nahezu aus.

- ***Portabel***
  Portabilität hat natürlich auch mit Architekturneutralität zu tun, geht jedoch noch darüber hinaus: Java kennt keine Unterschiede bei Datentypen auf verschiedenen Architekturen, was z.B. bei C++ oft zu Fehlern führen kann (unterschiedliche Bit-Größe für einen `int` oder die Problematik „Big Endian/Little Endian“).

- ***Interpretiert***
  Der Java-Interpreter (die JVM) kann Java-Bytecode direkt auf jeder Maschine ausführen, auf die der Interpreter portiert wurde.

- ***Schnell***
  Im Normalfall ist die Leistung von interpretiertem Bytecode schnell genug. Für besondere Situationen kann der Bytecode jedoch auch speziell für ein System in Maschinen-Code übersetzt werden, der als nativer Code natürlich noch schneller ist.

- ***Multithreaded***
  Im Gegensatz zu anderen Programmiersprachen ist es in Java relativ einfach, gleichzeitige Abläufe zu realisieren. Dies wird in Form von parallel und zeitgleich ablaufenden Threads verwirklicht, was vor allem der serverseitigen Programmierung entgegen kommt. Allerdings wird das Multithreading vom darunterliegenden Betriebssystem unter Umständen eingeschränkt, da Java das Multithreading an dieses überträgt.

- ***Dynamisch***
  In mancherlei Hinsicht ist Java dynamischer als C++. So können zum Beispiel problemlos Klassen um Methoden oder Eigenschaften erweitert werden, ohne dass dies ihre Klienten beeinflusst. Zudem können zur Laufzeit Informationen über Objekte ermittelt werden, was bei C++ gänzlich unmöglich ist.

Als Zusammenfassung ein weiteres Zitat aus [Java 2005]:

The Java language provides a powerful addition to the tools that programmers have at their disposal. Java makes programming easier because it is object-oriented and has automatic garbage collection. In addition, because compiled Java code is architecture-neutral, Java applications are ideal for a diverse environment like the Internet.

2.3 Betriebssysteme Linux und Windows

Ursprünglich war Windows nur eine grafische Erweiterung von MS-DOS (Microsoft-Disk Operating System). MS-DOS dominierte in den späten 1980er Jahren den Markt für Einzelplatzrechner. Windows als Graphical User Interface (GUI) benutzte DOS für alle Systemzugriffe. Anfänglich war nur kooperatives Multitasking[13] möglich, mit Windows 3.x auch präemptives Multitasking[14] zwischen DOS-Programmen und allen Windows-Programmen als Einheit betrachtet.

1995 startete die 32-Bit-Version von DOS mit Windows 95 als erstem Vertreter. Windows basiert nach wie vor auf MS-DOS, besitzt nun aber einen eigenen Systemkern (Kernel genannt). Windows 98 benutzte eingeschränktes präemptives Multitasking. Es schloss sich noch die Millennium Edition an (Ende 2000), die aber keine wesentlichen Neuerungen brachte.

Windows NT war ein echter Fortschritt. Es wartete mit präemptivem Multitasking und Speicherschutz auf und besaß einen eigenen Kernel. Speicherschutz bedeutet, dass Hardware nun nicht mehr wie früher direkt auf den Speicher zugreifen kann. Es folgte Windows 2000, das die professionelle NT-Linie für Unternehmen mit der Consumer-Variante vereinte und weitaus stabiler war als seine Vorgänger bis dato. 2001 erschien Windows XP, das die Verschmelzung beider Linien zum Abschluss brachte.
DiLog wurde unter Windows XP ohne Verwendung des 2004 herausgegebenen Service Pack 2 entwickelt.

Im Desktop-Bereich hat MS Windows quasi eine Monopolstellung inne. Das alternative Open-Source-Betriebssystem Linux schickt sich langsam an Microsofts Vormachtsstellung anzugreifen. Im Server-Bereich ist das UNIX-Derivat seit jeher sehr erfolgreich. Der Finne Linus Torvalds begann die Entwicklung und koordiniert auch heute noch die weltweit agierende Entwickler-Gemeinde. Genau genommen ist Linux kein Betriebssystem, sondern ein Betriebssystem-Kernel und wurde für X86-Prozessoren[15] geschrieben. Es existieren heute aber Portierungen für weitere Mikroprozessoren.

1994 erschien Version 1.0, Version 2.0 schloss sich im Juni 1996 an. Es folgten die Kernel-Versionen 2.2 (1999), 2.4 und die heute aktuelle Version 2.6.[16] Interessant ist hierbei, dass alle 2er-Versionen immer noch verbessert werden. Das scheint auch ein Vorteil von Open

[13] Beim kooperativen Multitasking sind die einzelnen Prozesse selbst dafür verantwortlich, die Kontrolle an den Betriebssystemkern zurückzugeben. Fehlerhafte Prozesse eines Programms können dabei das gesamte System zum Stillstand bringen.

[14] Präemptives Multitasking zeichnet sich durch höhere Stabilität aus. Der Betriebssystemkern teilt jedem Prozess eine gewisse Abarbeitungszeit zu, hält danach den Prozess an und startet einen anderen Prozess (Zeitscheiben-Strategie).

[15] Befehlssatz einer Mikroprozessor-Architektur der Firma Intel

[16] Freigegebene Kernel haben immer eine gerade Zahl hinter dem Punkt, Entwickler-Kernel eine ungerade.

Source zu sein. Der Erfolg von Linux ist neben Stabilität, Sicherheit und Erweiterbarkeit auch auf geringe Kosten zurückzuführen.[17]

Linux wird in sog. Linux-Distributionen an den Anwender ausgeliefert. Dabei handelt es sich um den Kernel, ergänzt um weitere Software, Installationshilfen und Handbücher. Es kann einfach nur Sinn machen, Software zu bündeln, viele Programme benötigen aber auch andere Programme um überhaupt zu funktionieren. Bekannte Distributionen sind Red Hat, SuSE, Mandrake, Debian oder Knoppix (von CD ohne Installation startbar). Jede Distribution zeichnet sich durch ein anderes Erscheinungsbild (Look & Feel) und unterschiedliche Installationsroutinen aus, alle beziehen sich jedoch auf denselben Quelltext.

2.4 Webserver Apache

Der Apache HTTP-Server ist – noch vor Microsofts Internet Information Server – der heutzutage gebräuchlichste und verbreitetste Webserver im Internet ([Apache 2005]).

> *Apache has been the most popular web server on the Internet since April of 1996. The October 2004 Netcraft Web Server Survey found that more than 67% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined.*

In der 1. Version, die am 1. Dezember 1995 herauskam, wurde der (produktive) Einsatz nur auf UNIX-Systemen empfohlen. Die noch nicht so lange am Markt befindliche Version 2 ist nun auch für Windows-(NT-)Systeme geeignet. Daneben läuft der Webserver auch auf vielen weiteren Betriebssystemen und gehört zur Kategorie der OSS.

Der Webserver ist modular aufgebaut, d.h. man kann Module für spezielle Aufgaben einbinden und die Server-Funktionalität dadurch erweitern. Beispielsweise ermöglicht mod_ssl die verschlüsselte Kommunikation zwischen Webserver und dem anfragenden Browser. Weitere Module können z.B. mit Hilfe von PHP oder Perl dynamische HTML-Seiten erstellen. Es besteht aber auch die Möglichkeit, das über die CGI-Schnittstelle zu erledigen.

Im einfachsten Fall liefert der Webserver statische HTML-Seiten an die anfragenden Webbrowser über HTTP/1.1 aus. In der DiLog-Konfiguration wird der Request über den JK-Connector an den Java-Applicationserver Tomcat weitergeleitet. Dieser leitet die generierte Ergebnisseite zurück an Apache und weiter an den anfragenden Browser.

2.5 Applicationserver Tomcat

Tomcat gehört dem Jakarta-Projekt an, ist ebenfalls kostenlose OSS und wird unter der Apache Software Licence vertrieben. Der Applicationserver Tomcat ist ein Servlet-Container, er implementiert die Spezifikationen Servlet und JSP von Sun Microsystems. Obwohl der Server auch „standalone“ arbeiten kann, erweitert er doch meist den Webserver Apache um

[17] Gerade Microsoft als erklärter Gegner von Open Source und Linux rechnet in Studien aber immer wieder vor, dass Linux durch die kompliziertere Administration und notwendige Schulungen von Personal im Endeffekt teurer sei.

die Servlet- und JSP-Funktionalität. Tomcat startete mit Version 3 und gelangte letztes Jahr bei 5.x an. Momentan werden durch Tomcat Servlets in Spezifikation 2.4 und JSP in Spezifikation 2.0 ermöglicht.

Der Jasper-Compiler wandelt Java Server Pages (JSP) in Servlets um. Dazu ist es notwendig, dass nicht nur das Java Runtime Environment (JRE) auf dem System installiert ist (das Tomcat zur eigenen Ausführung benötigt), es muss auch ein sog. Java 2 Software Development Kit (J2SDK) vorhanden sein. DiLog benutzt Tomcat in der Version 5.0.

2.6 JK-Connector

Das neue Tomcat/Apache-Plugin JK (mod_jk) ist der Nachfolger des früher verwendeten Apache-Moduls mod_jserv um die beiden Server miteinander zu verbinden. DiLog verwendet zur Zeit die überarbeitete Version von JK, die JK2 heißt. Mit JK2 wurde der native Part des Connectors überarbeitet und auch die Konfiguration erleichtert.

Im Gegensatz zu JServ, das nur auf Unix-Systemen funktionierte, läuft JK auf verschiedenen Betriebssystemen und unterstützt mehrere Webserver.[18] Zudem wird durch das Verwenden des neuen AJPv13-Protokolls verlässlichere SSL-Unterstützung (Secure Socket Layer) geboten. Alle Tomcat Engines von Version 3.2 bis 5.x können JK benutzen.

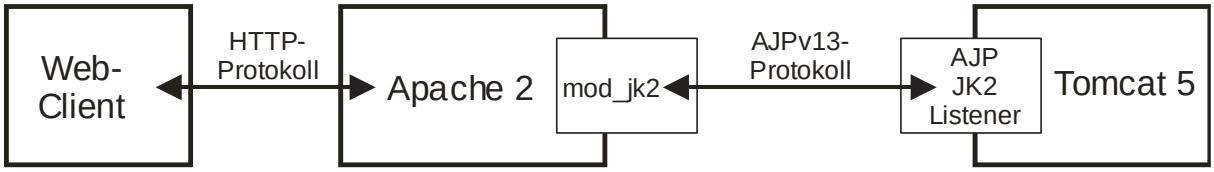


Abb. 2: Zusammenspiel der Webserver-Komponenten

Beim Refactoring von JK zu JK2 wurden die Listener[19] auf Seiten des Servlet-Containers nicht verändert. Trotzdem gibt es Unterschiede in den Konfigurationsdateien, vor allem auf Seiten des Webservers. JK2 funktioniert zwar auch mit Apache 1.3, wurde jedoch für Apache 2 konzipiert. Verbesserte Modularität und Unterstützung schneller UNIX-Sockets zeichnet JK2 aus. Die weiteren Unterschiede können detailliert auf [JK2 alt] nachgelesen werden.

Mit Nachricht am 15. November 2004 wurde die Weiterentwicklung des JK2-Connectors leider überraschend eingestellt. Diese Entscheidung ist auf mangelndes Interesse durch Entwickler zurückzuführen. Als weiterer Grund wird die Komplexität von JK2 angeführt: „Other reason was lack of users interest in adopting JK2, caused by configuration complexity when compared to JK." ([JK2 neu]). Dies widerspricht allerdings den Aussagen zu JK2 gegenüber JK auf ([JK2 alt]): „… the configuration has been simplified a lot."!

DiLog verwendet – da einmal eingerichtet und dokumentiert – weiterhin JK2 als Verbindung zwischen Apache und Tomcat. Demzufolge wird in Kapitel 3 auch ausschließlich auf die Konfiguration in Bezug auf JK2 eingegangen.

[18] Webserver wie IIS, Netscape/iPlanet und Domino können JK genauso wie Apache benutzen.

[19] Der Listener (Zuhörer) entstammt dem Beobachter-Entwurfsmuster. Er „lauscht" auf alle für ihn relevanten Ereignisse, Nachrichten oder Informationen und stößt die Weiterverarbeitung durch Tomcat an.

Im Moment wird also nur noch JK weiterentwickelt, es kam bereits eine neue Version 1.2.8 nach Einstellung der Entwicklungsarbeit an JK2 auf den Markt. Sollte sich in Zukunft zeigen, dass JK2 sicherheitsrelevante Fehler enthält, so muss evtl. auf JK umgestiegen werden.

2.7 Verschlüsselung SSL

Mit Hilfe von Secure Socket Layer (SSL) lassen sich Daten verschlüsselt übertragen. Das Übertragungsprotokoll setzt sich in der Transportschicht (Schicht 4) des OSI-Netzwerkmodells[20] auf TCP (Transmission Control Protocol). Ursprünglich wurde SSL von Netscape entwickelt, mischte sich dann aber mit einem (besseren) Konkurrenzprodukt von Microsoft. Mit der Standardisierung von SSL im Jahre 1999 wurde es zu Transport Layer Security (TLS) umbenannt. SSL in der Version 3.0 entspricht mit nur kleinen Unterschieden TLS 1. So meldet sich TLS 1.0 im Header als SSL-Version 3.1.

Kurz zur Funktionsweise: Das SSL Record Protocol liegt in der untersten SSL-Ebene und kapselt die höheren Protokolle (Ebenen). Beispiele dafür sind das Hypertext Transfer Protocol (HTTP) oder das SSL Handshake Protocol, welches zur Authentifizierung von Server und Client oder zur Vereinbarung des zu benutzenden Verschlüsselungsverfahrens dient. Authentifizierung und nachfolgende Kommunikation laufen wie folgt ab:

- Client (z.B. Webbrowser) sendet Verbindungsanfrage an Server.
- Server antwortet und sendet ein Zertifikat.
- Client versucht das Zertifikat zu authentifizieren. Dabei stellt dieses Zertifikat den öffentlichen Schlüssel des Servers dar. Bei Misserfolg bricht der Client nach Bestätigung des Benutzers die Verbindung ab (Abb. 3).
- Bei erfolgreicher Authentifizierung erstellt der Client das sog. *pre-master secret*, verschlüsselt dieses mit dem öffentlichen Schlüssel des Servers und schickt es an diesen zurück. Zusätzlich erzeugt der Client daraus das *master secret*.
- Der Server entschlüsselt das *pre-master secret* mit seinem privaten Schlüssel und erstellt ebenfalls das *master secret*.
- Nun erstellen sowohl Server als auch Client den *session key* aus dem *master secret*. Der Sitzungsschlüssel ist ein symmetrischer Schlüssel und dient nun zum gesicherten Austausch von Daten. Dies ist insofern von Vorteil, weil die symmetrische Verschlüsselung schneller ist als die (einmalige) asymmetrische Verschlüsselung zu Beginn des Datenaustausches zwischen Client und Server.

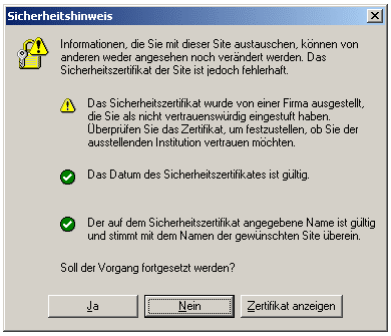


Abb. 3: Unstimmigkeiten bei Prüfung des Zertifikats (hier IE)

Die gebräuchlichste und häufigste Form der Verwendung von SSL-Verschlüsselung ist heutzutage das Protokoll Hypertext Transfer Protocol Secure (HTTPS). Dabei wird vor allem

[20] Das standardisierte OSI-Modell (Open Systems Interconnection Reference Model) ist die Grundlage für viele Netzwerkprotokolle und regelt den Transport von Daten in Netzwerken.

SSL 3.0/TLS mit der Verschlüsselungstechnik RSA[21] oder AES[22] eingesetzt.

Die Server-Zertifikate werden von einer allgemein anerkannten Zertifizierungsstelle – einer sog. Certification Authority (CA) – ausgestellt und sind deshalb als vertrauenswürdig einzustufen. Das ist jedoch mit nicht unerheblichen Kosten verbunden. Deshalb benutzt DiLog selbstsignierte Zertifikate für die sichere Datenkommunikation und für das Chat-Applet. Prinzipiell geht es nicht vorrangig um die Authentifizierung des Servers, sondern um die verschlüsselte und sichere Datenübertragung von persönlichen Daten, die auch ohne anerkanntes Zertifikat sichergestellt ist. Die dadurch resultierende Fehlermeldung ist in Abb. 3 für den Internet Explorer dargestellt (vorausgesetzt der Benutzer hat das Zertifikat nicht früher schon einmal als vertrauenswürdig eingestuft).

2.8 Datenbank MySQL

MySQL ist eine relationale SQL[23]-Datenbank, auch als sog. RDBMS (Relational Database Management System) bezeichnet. Ganz genau genommen stimmt das nicht [Reese 2003] (S. 4):

> *MySQL an sich ist keine Datenbank. Vielmehr handelt es sich hier um Software, die es Ihnen ermöglicht, elektronische Datenbanken anzulegen, zu pflegen und zu verwalten. Diese Art von Software ist als Datenbank-Management-System (DBMS) bekannt. Ein DBMS fungiert als »Makler« (Broker) zwischen der physikalischen Datenbank und den Benutzern dieser Datenbank.*

1994 entwickelte Michael Widenius MySQL für die schwedische Firma TcX (welche die meiste Zeit nur aus ihm selbst bestand). Ein Jahr später war die erste Version fertig, die ihren Einsatz zusammen mit webbasierten Anwendungen fand. Schon bald danach wurde MySQL als OSS der Öffentlichkeit zur Verfügung gestellt. Nach und nach wurde das System auf verschiedene Betriebssysteme portiert, wie z.B. Linux, Windows, MacOS oder OS/2. Nach Schätzungen von MySQL AB soll MySQL weltweit auf ca. 4 Millionen Servern laufen (siehe [Reese 2003] (S. 7)).

Vor ein paar Jahren hat sich TcX in die Firma MySQL AB umgewandelt und 2003 die Datenbankaktivitäten der Firma SAP übernommen. Die SAP DB wurde in MaxDB umbenannt und bietet alles, was eine vollwertige professionelle Datenbank bieten muss. Allerdings ist das System auch komplexer und umfangreicher als MySQL.

Der Erfolg von MySQL ist maßgeblich auf seine hohe Geschwindigkeit und geringe Komplexität zurückzuführen, denn MySQL bietet nicht alle Anforderungen, die

[21] RSA ist ein asymmetrisches Verschlüsselungsverfahren, d.h. es verwendet verschiedene Schlüssel zum Ver- und Entschlüsseln. Es ist nach seinen Erfindern Ronald **R**ivest, Adi **S**hamir und Leonard **A**dleman benannt.

[22] AES ist ein symmetrisches Verschlüsselungsverfahren und seit Oktober 2000 der Nachfolger von DES (Data Encryption Standard). DES war aufgrund seiner zu kurzen Bit-Länge unsicher geworden und konnte 1998 mittels Brute-Force-Attacke (Erraten/ Ausprobieren von Schlüsseln) geknackt werden.

[23] SQL (Structured Query Language) ist ein Quasi-Standard zum Manipulieren von Daten in einer Datenbank. Leider wurde der Standard-Sprachumfang durch eine Reihe von SQL-Datenbanken herstellerspezifisch erweitert, was den Wechsel einer in einem System eingesetzten Datenbank nicht gerade erleichtert.

normalerweise an eine Datenbank gestellt werden. So sind beispielsweise Transaktionen (Zusammenfassung mehrerer SQL-Befehle und „Undo“-Funktion bei Scheitern einer Anweisung) und Foreign Key Constraints (Sicherstellung von Datenintegrität über mehrere Tabellen hinweg) noch nicht lange und auch nur bei Verwendung eines zusätzlichen Tabellentyps möglich.

DiLog verwendet Transaktionen und demzufolge den Tabellentyp InnoDB, was durch den Zusatz `TYPE=InnoDB` bei jeder Create-Anweisung zur Erzeugung der einzelnen Tabellen erreicht wird.
Die Administration kann (umständlich) mit den mitgelieferten Programmen mysql und mysqladmin erfolgen. Es gibt aber auch wesentlich einfacher zu benutzende Programme mit GUI, wie z.B. phpMyAdmin (PHP Installation notwendig), MySQL Administrator zusammen mit MySQL Query Browser (von MySQL AB selbst) und MySQL-Front. Diese Werkzeuge erlauben nicht nur Administration, sondern sind auch komfortabel zum Absetzen von SQL-Anweisungen und zur Darstellung von SQL-Abfragen verwendbar.

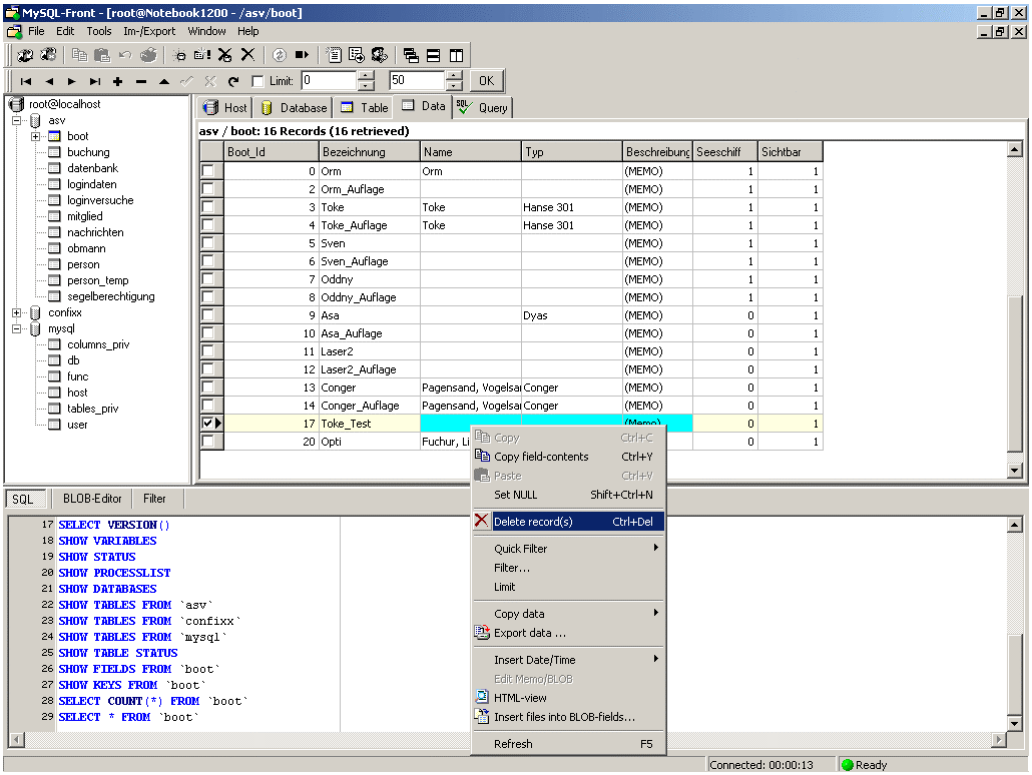


Abb. 4: SQL-Werkzeug MySQL-Front mit Darstellung und Bearbeitung der Relation Boot

3 Software-Entwicklung und Hardware

Dieses Kapitel zeigt die für die Entwicklung von DiLog nötig gewesenen Systeme, deren Einrichtung und Konfiguration auf. Für das Aufsetzen der Server existieren separate PDF-Dateien, die in aller Ausführlichkeit Schritt für Schritt alle Vorgänge (auch für Linux-Einsteiger verständlich[24]) zu allen notwendigen Installationen beschreiben. Diese PDF-Dateien sind auf der mit diesem Schriftstück ausgelieferten CD-ROM enthalten.

3.1 Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus einer normalen Windows-Installation (Windows XP Home). Als integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) für Java kam Eclipse zum Einsatz. Reine Textdateien wie Konfigurationsdateien (Struts-Konfigurationsdateien, XML-Dateien, Server-Konfigurationsdateien), HTML-Dateien, CSS-Dateien oder JSP-Dateien wurden mit dem Texteditor UltraEdit erstellt. UML-Diagramme machte die Verwendung von Together möglich. Es zeigte sich, dass 512 MB Arbeitsspeicher (und ein Intel-Pentium-3-Prozessor mit 1200 MHz Taktung) lange nicht für ein flüssiges Arbeitstempo ausreichen, vor allem wenn alle Java-Programme (Eclipse, Tomcat, Together, IETool) und andere notwendige Software (Apache, MySQL, MySQL-Front, MS Access, UltraEdit, Mail-Client, Acrobat Reader) zusammen ausgeführt werden mussten.

3.1.1 Installation und Konfiguration

Nach einer normalen Windows-Installation folgte eine normale Java-Installation (J2SDK 1.4.2). Es kann zweckmäßig sein, das `bin`-Verzeichnis der Java-Installation in die Windows-Umgebungsvariable `PATH` zu schreiben. Die Installation von Eclipse verursachte auch keine größeren Probleme, lediglich wollte die IDE nach Ausführen der Datei `eclipse.exe` nicht starten. Die Angabe des J2SDKs und weiterer Parameter als Startparameter der EXE-Datei behob das Problem (`E:\winXP\eclipse\eclipse.exe -vm E:\winXP\java\jsdk142_04\bin\javaw.exe -Xms100m -Xmx200m -data D:\eclipse`).

Zur Zeit der Installation war auf der Apache-Website kein Installationspaket von Apache 2 für Windows mit integriertem SSL-Modul zu finden. Die einzige Möglichkeit schien die einer Kompilierung mit C++ unter Windows. Glücklicherweise ließ sich auf einer Website im Netz doch noch eine Installation finden, die ein Apache-User mit demselben Problem tatsächlich kompiliert hatte. Für die Produktionsumgebung könnte es durchaus gefährlich sein, solch ein Software-Paket aus unsicherer Quelle zu verwenden, schließlich könnte die Apache-Software verändert worden sein und nun als Trojaner o.ä. fungieren. Jedoch schien das Risiko für die Entwicklungsumgebung kalkulierbar (keine Ausfallgefahr, kein Datenverlust oder Preisgabe von persönlichen Benutzerdaten an Unbekannte).

[24] Zu Beginn des Projekts war ich völlig unerfahren, was die Installation und Administration von Linux und seinen Komponenten anging. Nicht verfügbare Binärdateien erforderten Kompilierung und Konfiguration von Source Code mit einem C-Compiler und zwangen mich, tiefer als gewollt in die Materie einzusteigen.

Nach der Installation sollte der Webserver als automatischer Windows-Service registriert werden (`apache -k install`) oder jedes Mal per Hand mit `net start apache2` gestartet werden. Mit der URL (Uniform Resource Locator) `http://localhost` kann die Funktion des installierten Webservers überprüft werden.

Tomcat kann ebenfalls bequem durch Ausführen der entsprechenden EXE-Datei installiert werden. Bei der Installation muss der Pfad zu einem gültigen J2SDK und die Kennung für den Administrator angegeben werden. Wenn Tomcat richtig installiert wurde[25], so sollte der Aufruf von `http://localhost:8080` im Browser die Tomcat-Website anzeigen und die dort aufgeführten JSP- und Servlet-Beispiele sollten ebenfalls funktionieren.
Anschließend muss der Tomcat-Manager eingerichtet werden. Dazu muss die Datei `/tomcat5/conf/tomcat-users.xml` editiert werden. Danach soll die Datei den Eintrag `<user username="root" password="passw" roles="standard, manager, admin"/>` enthalten. Jetzt sollte die erfolgreiche Anmeldung am mitinstallierten Tomcat-Manager auf `http://localhost:8080/manager/html` möglich sein.
Hinweis: Zu Tomcat 4.0 merkt [Roßbach 2001] folgendes an:

> *Der Manager ist in der Lage, alle Webapplikationen zu stoppen, zu starten, erneut zu laden und eine Anwendung zu löschen. Die Anwendung wird dabei nicht physikalisch gelöscht und ist nach dem Neustart des Servers wieder verfügbar.*

Dies wurde in Tomcat 5.0 geändert. Ein Mausklick auf „Löschen“ entfernt die Anwendung unwiederbringlich und ohne vorherige Nachfrage!

Zur Einbindung des JK2-Connectors wird die Datei `mod_jk2.dll` nach `apache2/modules` kopiert. Die Datei `tomcat5/conf/jk2.properties` muss, wie der Pfad andeutet, im Tomcat-Konfigurationsverzeichnis abgelegt werden. Danach müssen mindestens zwei Dateien angepasst werden:

- In Datei `apache2/conf/httpd.conf` wird Apache angewiesen, das Verbindungsmodul zu laden (`LoadModule jk2_module modules/mod_jk2.dll`).
- Die Datei `apache2/conf/workers2.properties` ist in Listing 1 abgebildet.

```
# only at beginnin. In production uncomment it out (oder WARN)
[logger.apache2]
level=ERROR

[shm]
file=e:\server\apache2\logs\shm.file
size=1048576

# Example socket channel, override port and host.
[channel.socket:localhost:8009]
port=8009
host=localhost
info=Ajp13 forwarding over socket

# define the worker
[ajp13:localhost:8009]
channel=channel.socket:localhost:8009
```

Listing 1: Apache-Konfigurationsdatei workers2.properties

[25] Eine alte deinstallierte Version von Tomcat 4 verhinderte die ordnungsgemäße Installation von Tomcat 5 zunächst (Absturz bei der Installation). Erst als der nicht deinstallierte aber immerhin deaktivierte (!) Service der alten Version manuell entfernt war, konnte eine neue Installation von Tomcat 5 gelingen.

```
[uri:/jsp-examples/*]
worker=ajp13:localhost:8009
info=Die JSP-Beispiel-Seiten

[uri:/servlets-examples/*]
worker=ajp13:localhost:8009
info=Die Servlet-Beispiel-Seiten

[uri:/manager/*]
worker=ajp13:localhost:8009
info=Tomcat-Manager

[uri:/admin/*]
worker=ajp13:localhost:8009
info=Tomcat-Admin

[uri:/hello/*]
worker=ajp13:localhost:8009
info=Hallo-Welt mit Servlet und JSP (eigenes Custom Tag)

[uri:/dilog/*]
worker=ajp13:localhost:8009
info=DiLog - Digitales Logbuch
```

Listing 1: Apache-Konfigurationsdatei workers2.properties (Fortsetzung)

- Wenn Tomcat nicht mehr standardmäßig über Port 8080 angesprochen werden soll, gilt es in `/tomcat5/conf/server.xml` den Standard-Connector auszukommentieren:
  ```
  <!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
  <!-- Connector [..] /-->
  ```
- Wenn der Standard-Port von JK2 (8009) geändert werden soll, muss dieser auch in Tomcats Dateien `server.xml` und `jk2.properties` geändert werden.

Nach einem Neustart von Apache und Tomcat (evtl. Reihenfolge beachten) müsste der Aufruf der URL `http://localhost/jsp-examples` (ohne Portangabe) nun die Beispielseite für JSPs laden. Der Zugriff auf Tomcat erfolgt nun über Apache.

MySQL verfügt ebenfalls über eine ausführbare Installationsdatei. Danach müssen ggf. Einstellungen der Datei `my.ini` im Windows-Verzeichnis angepasst werden (siehe Listing 2).

```
[mysqld]
basedir=E:/winXP/mysql4
#bind-address=192.168.1.229
#datadir=E:/winXP/mysql4/data
datadir=D:/da_asv_hh/diplomarbeitsmaterial/db-design/mysqldata
#language=E:/winXP/mysql4/share/your language directory
#slow query log#=
tmpdir=D:/da_asv_hh/diplomarbeitsmaterial/db-design/mysqldata/temp
port=3306
#set-variable=key_buffer=16M
[WinMySQLadmin]
Server=E:/winXP/mysql4/bin/mysqld.exe
```

Listing 2: MySQL-Konfigurationsdatei my.ini

Die wichtigsten Einstellungen betreffen wohl die Portnummer und das Tabellenverzeichnis (datadir), in dem MySQL alle Tabellen als Dateien und zusätzliche Verwaltungsdateien ablegt.
Mit `net start mysql` wird der Service gestartet (falls nicht automatisiert). Es folgt die wichtige Rechte- und Benutzerverwaltung:

- Als Administrator auf der Konsole mit `mysql.exe` oder einem grafischen Frontend einloggen: `mysql -h localhost -P 3306 -u root` (ohne Passwort).

- Aus Sicherheitsgründen wird dringend angeraten, in der Tabelle `user` der Datenbank `mysql` die Einträge ohne Name und Passwort zu löschen.
- Für DiLog müssen zwei neue Benutzer angelegt werden: Benutzer asv_mysql hat beschränkte Rechte, asv_createdb dagegen uneingeschränkte Rechte, da dieser Zugang benutzt wird, um einmalig die Datenbank zu erstellen. Nähere Informationen zur SQL-Syntax und zur `PASSWORD()`-Funktion sind im begleitenden PDF „Vorgehen WinXP-Installation“ enthalten.
- Dann wird der Administrator-Zugang passwortgeschützt, indem auf der Konsole `mysqladmin.exe` ausgeführt wird: `mysqladmin -u root password testpwd`. Wenn auf die Datenbank remote von entfernten Rechnern zugegriffen werden soll, muss für Feld „Host“ in Tabelle `user` das Zeichen % für Zugriff von beliebigen Rechnern oder eine eingeschränktere Rechnerauswahl eingegeben werden.

Damit die Benutzer- und Rechteänderungen wirksam werden können, muss die Datenbank gestoppt und neu gestartet werden.

3.2 Testumgebung

Für die Testumgebung wird ein alter PC mit einem AMD-K6-2-Prozessor (500 MHz) mit 512 MB Arbeitsspeicher benutzt. Dies ist für die Linux-Distribution Red Hat 7.3 (Valhalla, Kernel 2.4.18) und den Serverbetrieb ohne Benutzeroberfläche völlig ausreichend. Im Gegensatz zur Produktionsumgebung ist allerdings ein mehr oder weniger genutzter X-Server zur Anzeige einer grafischen Benutzeroberfläche installiert, was für die Verwendung bestimmter grafischer Java-Methoden von Bedeutung ist.
Generell war es das Ziel, die Testumgebung der Produktionsumgebung möglichst weit anzugleichen, um etwaiges Fehlverhalten bei DiLog-Updates schon vor der Installation auf der Produktionumgebung aufzuspüren.

3.2.1 Installation und Konfiguration

Prinzipiell ist die Installation unter Linux ähnlich wie unter Windows, nur viel komplizierter und umfangreicher. Dem Leser und Software-Entwickler, der vor einer ähnlichen Installation steht, wird auch (und vor allem) hier das entsprechende PDF auf der CD-ROM ans Herz gelegt („Vorgehen Linux-Installation“). Dieses Unterkapitel geht wegen der Fülle an Befehlen nur in Sonderfällen auf die einzelnen, ständig wiederkehrenden Anweisungen wie `gunzip`, `tar`, `configure`, `make`, `rpm` und ihre vielfältigen Parameter ein.

Da der alte Rechner nicht über CD booten kann, musste mit RaWrite unter MS-DOS erst eine Boot-Diskette von einem Image auf CD erstellt werden. Die reine Linux-Installation gestaltete sich erstaunlich einfach, es wurden alle Geräte (selbst der SCSI-Controller) erkannt. Anschließend wurde ein reiner Konsolen-Editor mit dem Namen Joe installiert. Es wurde auch eine Installation von OpenSSL 0.9.7d notwendig, um dem Webserver die Benutzung von SSL zu ermöglichen.

Die MySQL-Installation erfolgte mit Hilfe des Red Hat Package Manager (RPM-System). Mit `rpm -Uvh MySQL-server-4.0.17.rpm` und `rpm -Uvh MySQL-client-4.0.17.rpm` sind Server und Client installiert. Äquivalent zur Windows-Installation wird der Administrator mit einem Passwort versehen: `/usr/bin/mysqladmin -u root password testpwd`. Desgleichen werden unnötige Benutzerzugänge gelöscht und neue Benutzer eingefügt. Für das Feld Host in Tabelle user kann der Wert 192.168.1.% (Zugriff nur aus dem lokalen Netzwerk) eingetragen werden. Nach einem Neustart von MySQL sind alle Änderungen übernommen. Unter Linux speichert MySQL seine Tabellen unter `/var/lib/mysql`.

Bevor Apache 2.0.48 mit SSL-Unterstützung installiert werden konnte, musste die alte Apache 1.3-Version deinstalliert werden. Mit der Konfiguration muss entschieden werden, ob das Modul mod_ssl als Dynamic Shared Object (DSO) oder statisch eingebunden werden soll. Hier wird das Modul dynamisch eingebunden: `./configure --enable-ssl=shared`. Wenn die Konfiguration oder der Make[26]-Durchlauf mit Fehlermeldungen abbrechen, sollte man mit den zusätzlichen Parametern `--enable-so` und `-with-ssl=/usr/local/ssl` experimentieren. Letzterer gibt den Pfad zur OpenSSL-Installation an.

Bei der Registrierung von Apache als automatisch startenden Service gab es eine weitere Überraschung. Ein Daemon[27] Script (Datei `httpd`) wurde nach `/etc/rc.d/init.d` kopiert und ausführbar gemacht (`chmod a+r+x+ httpd`). Die eigentliche Registrierung mit dem Aufruf `/sbin/chkconfig --add httpd` wurde mit der Meldung „bad interpreter: no such file or directory“ quittiert. Des Rätsels Lösung: Mit dem Programm `dos2unix -o httpd` lässt sich die Datei in das richtige Format bringen.

Bevor Tomcat installiert werden kann, sollte Java installiert sein. Die ZIP-Datei von Sun Microsystems wird mit `gunzip` entpackt, ausführbar gemacht (`chmod a+x j2sdk-1_4_2_03-linux-i586.bin`) und gestartet (`./j2sdk-1_4_2_03-linux-i586.bin`). Um Java systemweit zur Verfügung zu stellen, müssen in der Datei `/etc/profile` vor der Zeile `export PATH` die beiden Zeilen `PATH="$PATH:/usr/java/j2sdk1.4.2_03/bin"` und `export JAVA_HOME=/usr/java/j2sdk1.4.2_03` eingetragen werden.
Es hat alles geklappt, wenn die Eingabe von `java -version` nun die korrekte Java-Version anzeigt (evtl. ist ein Neustart notwendig, um die eben gesetzten Variablen zu exportieren).

Nach Dekomprimierung und Entpacken wird das Tomcat-Verzeichnis nach `/usr/local/` verschoben und in `tomcat5` umbenannt. `groupadd tomcat` und `useradd -g tomcat tomcat` legen einen Account für Tomcat an, dem das Tomcat-Verzeichnis zugewiesen wird (`chown tomcat.tomcat /usr/local/tomcat5`). Um auch Tomcat bei jedem Systemstart automatisch zu starten wird das Daemon Script mit Namen `tomcatd` nach `/etc/rc.d/init.d` kopiert, ausführbar gemacht und registriert. Im „Teufelsskript“ muss eine

[26] Make liest ein sog. Makefile, in dem der Übersetzungsprozess eines Programms formal erfasst ist. Diese Formalisierung beschreibt, welche Quelltext-Dateien der Compiler zu welchen Objektdateien verarbeitet und welche Objektdateien vom Linker dann zu Programmbibliotheken oder ausführbaren Programmen verbunden werden. Alle Schritte erfolgen unter Beachtung von Dateiabhängigkeiten. Leider funktioniert das nicht immer korrekt: Es gab einige fehlerbedingte Abbrüche, die entstehen können, wenn die Bibliotheken und Dateien nicht kompatibel zueinander sind oder das Makefile nicht dazu passt.

[27] Als Daemon (engl. Dämon, Teufel, Arbeitstier) bezeichnet man unter Linux, bzw. UNIX ein Programm, das (meistens ab Systemstart) „unsichtbar“ im Hintergrund abläuft und bestimmte Dienste zur Verfügung stellt. Dies entspricht dem Terminus Dienst oder Service bei Windows-Systemen.

Umgebungsvariable gesetzt werden, wenn kein X-Server installiert ist: `export CATALINA_OPTS="-Djava.awt.headless=true"` verhindert, dass bei Fehlen des X-Servers manche AWT-Klassen Exceptions werfen. Anschließend wird noch der Tomcat-Manager, wie zuvor schon aufgezeigt, eingerichtet.

Die Installation des JK2-Connector läuft prinzipiell genau wie die Installation unter Windows ab, wartete aber mit einer unangenehmen Überraschung auf. Das Verbindungsmodul war als kompiliertes Binary auf der Apache-Website nur für das Betriebssystem Solaris und nicht für Intel-x86-Systeme zu haben. Beim Start von Apache meldete dieser folgenden Fehler: „ELF file data encoding not little-endian“. ELF bedeutet Executable and Linkable Format und beschreibt das Standard-Binärformat von ausführbaren Programmen.

Exkurs: Big-Endian vs. Little-Endian

Bei Computern gibt es mehrere Möglichkeiten, wie aus mehreren Bytes bestehende Zahlenwerte im Speicher abgelegt werden können: Neben den wichtigen Vertretern Big Endian (BE) und Little Endian (LE) gibt es bei älteren Systemen auch noch Middle Endian. Der Unterschied zwischen BE und LE lässt sich am besten an einem Beispiel deutlich machen. Es wird der 32-Bit-Integer-Wert mit der hexadezimalen Darstellung 0xB1C2D3E4 wiedergegeben.

Man spricht von LE, wenn die 4 Bytes (= 32 Bit) in der Reihenfolge E4D3C2B1, also das niederwertigste Byte an der niedrigsten Speicheradresse, abgelegt werden. Im Gegensatz dazu wird das Ablegen des höchstwertigen Bytes an der niedrigsten Speicheradresse – also B1C2D3E4 – als BE bezeichnet. Wenn man so will, entspricht die normale Darstellung von Zahlen in der Leserichtung der westlichen Welt dem Format Big Endian.

Beide Systeme haben Vorteile. So können bei Verwendung von LE mathematische Routinen für Berechnungen einfach geschrieben werden, während bei BE das Vorzeichen eines Zahlenwertes einfach zu prüfen ist oder auch Umrechnungen von binär zu dezimal einfach zu realisieren sind. Netzwerk-Protokolle legen normalerweise die Byte-Reihenfolge immer fest, um fehlerlosen Datenaustausch zu ermöglichen. Im Falle von TCP/IP wird die BE-Darstellung verwendet. Netzwerktreiber von PCs müssen also eine „Umsortierung" veranlassen.

Das Thema Byte-Reihenfolge ist auch unter dem Namen NUXI-Problem bekannt: Das Wort UNIX (2 Bytes) kann je nach Reihenfolge in LE als NUXI und in BE als UNIX vorliegen. Ihren Ursprung haben die Bezeichnungen witzigerweise in J. Swifts Roman „Gullivers Reisen": Die Bewohner von Liliput sind in zwei verfeindete Lager gespalten (die „Little-Endians" und „Big-Endians"), da sie ständig darüber in Streit geraten, wie denn ein Ei aufzuschlagen sei - am spitzen oder am dickeren Ende.

Suns Betriebssystem Solaris[28] ist zwar wie Linux auch ein UNIX-Betriebssystem, jedoch arbeiten die Mainframe[29]-Systeme unter Solaris im Gegensatz zu PC-Systemen mit der Byte-Reihenfolge Big Endian. So blieb als einzige Lösung das Modul selbst zu kompilieren…

[28] Randbemerkung: Solaris bedeutet im Lateinischen dasselbe wie Sun im Englischen. Damit konnte die Firma Sun das aus dem ursprünglichen Betriebssystem SunOS hervorgegangene neue Betriebssystem in Solaris umbenennen ohne seine Bedeutung und die Anlehnung an den Firmennamen zu ändern.

Nach gunzip und tar wird ins Verzeichnis `jakarta-tomcat-connectors-jk2-2.0.2-src/jk/native2` der entpackten Baumstruktur gewechselt und die Software mit `./configure --with-apxs2=/usr/local/apache2/bin/apxs --with-tomcat41=/usr/local/tomcat5 --with-java-home=/usr/java/j2sdk1.4.2_03 --with-jni --with-pcre` konfiguriert. Falls sich dabei sofort oder beim anschließenden Kompilieren Probleme ergeben, sollte man die Flags `--with-jni` oder `--with-pcre` weglassen. Als Ergebnis der Kompilierung befinden sich die Dateien `mod_jk2.so` und `jkjni.so` im Verzeichnis `jakarta-tomcat-connectors-jk2-2.0.2-src/jk/build/jk2/apache2`. Wirklich gebraucht wird nur die erste Datei, welche ins Apache-Modulverzeichnis kopiert wird. Damit Apache das Modul laden kann, muss in der Konfigurationsdatei `httpd.conf` der Eintrag `LoadModule jk2_module modules/mod_jk2.so` gemacht werden.

3.3 Produktionsumgebung

Obwohl DiLog noch nicht im Einsatz ist, wurde als Produktionsumgebung der virtuelle Server[30] (vServer) der Firma BSB Service GmbH (Tochter der intergenia AG) gesetzt. Für 10 € pro Monat bekommt man einen virtuellen Linux-Server mit Root-Zugang, fester IP-Adresse, 2 GB Webspace, 50 GB Traffic pro Monat und einer de-Domain. Neuerdings hat man sogar die Auswahl zwischen Red Hat 9, Fedora Core 1, SuSE Linux 9 und Debian 3.1. Auf dem vServer ist kein X-Window-System (X-Server) installiert. Deshalb erfolgt die Installation und Konfiguration von Zusatzsoftware ausschließlich über die Konsole. Verschlüsselten Zugang per SSL ermöglichen Programme wie Putty und WinSCP2.

3.3.1 Installation und Konfiguration

Die Installation des Produktivsystems lehnt sich eng an die Installation des Testsystems an. Deshalb werden hier nur noch abweichende Vorgänge erwähnt. Diese kommen u.a. durch unterschiedliche Versionen des vorinstallierten Betriebssystems Linux und des Webservers Apache oder vorhandene Zusatzsoftware wie Confixx PREMIUM EDITION (Serververwaltung über Webinterface) zustande.

MySQL ließ sich nicht mit RPM installieren, bzw. updaten, deshalb war es nötig, auf die etwas unbequemere Verwendung einer .tar.gz-Datei zurückzugreifen. Ähnlich wie bei Tomcat wird auch für MySQL ein separater Benutzeraccount angelegt (`groupadd mysql`, `useradd -g mysql mysql`). Nach dem Umkopieren (`cp -R /temp/mysql-standard-4.0.16-pc-linux-i686 /usr/local/mysql-standard-4.0.16`) wird im Verzeichnis `/usr/local` ein symbolischer Link gesetzt (`ln -s mysql-standard-4.0.16 mysql`).

---

[29] Mainframes sind Großrechner, die weit über die Kapazitäten von Personal Computern hinausreichen. Mainframe-Systeme zeichnen sich durch hohe Zuverlässigkeit und Ein-Ausgabe-Leistung aus. Die genau aufeinander abgestimmten Hardware-Komponenten sind hochgradig redundant und robust, sodass sie sogar im laufenden Betrieb ausgetauscht oder aufgerüstet werden können.

[30] Ein virtueller Server simuliert für eine bestimmte Anzahl an Benutzer (hier 250) einen Root-Server mit vollem Zugang, unbegrenzten Installationsmöglichkeiten und der Möglichkeit des Neustarts.

Nun muss im Verzeichnis `/mysql/scripts/ mysql_install_db` gestartet werden. Es schließen sich die Befehle `chown -R root /usr/local/mysql`, `chgrp -R mysql /usr/local/mysql` und `chown -R mysql /usr/local/mysql/data` an.
Es wird nun ein Start- bzw. Stopp-Skript an die entsprechende Stelle im Linux-Filesystem kopiert und Verlinkungen hergestellt:

- `cp /usr/local/mysql/support-files/mysql.server /etc/rc.d/init.d`
- `ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc3.d/S99mysql`
- `ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc0.d/S01mysql`

Die Datei `/etc/my.cnf` benötigt folgende Einträge:

```
[mysql.server]
basedir=/usr/local/mysql
socket=/var/lib/mysql/mysql.sock
[client]
socket=/var/lib/mysql/mysql.sock
```

Die weiteren Arbeitsschritte zur Installation von MySQL 4.0.16 (Standard) sind identisch mit der Installation auf dem Testsystem.

Apache 2.0.49 lässt sich recht ähnlich wie in der Testumgebung installieren. Lediglich die auszuführenden vorbereitenden Schritte sind anders: In die Datei `/etc/ld.so.conf` muss `/usr/local/ssl/lib` eingetragen werden. Danach die Dateien `libcrypto.so.0.9.7` und `libssl.0.9.7` aus Verzeichnis `/usr/local/ssl/lib` nach `/lib` kopieren. Nachdem der Konfigurierungsvorgang (`./configure --enable-ssl=shared --enable-so -with-ssl=/usr/local/ssl --enable-suexec=shared --enable-info=shared --enable-cgi=shared`) beendet ist, verläuft die Installation identisch mit der Testumgebung.

Confixx benötigt eine PHP-Installation, die erst nach MySQL erfolgen kann, da beim Konfigurierungsvorgang auf die MySQL-Datenbank verwiesen wird: `./configure --with-openssl=/usr/local/ssl --with-mysql= /usr/local/mysql --with-apxs2=/usr/ local/apache2/bin/apxs --with-zlib`. In der Apache-Konfigurationsdatei `AddType application/x-httpd-php .php .phtml` hinzufügen, fertig.

3.4 Konfiguration

In den vorhergehenden Unterkapiteln wurde die grundlegende, generell notwendige Konfiguration erläutert. Dieses Unterkapitel beschäftigt sich hauptsächlich mit weiterführender Konfiguration, geht aber im Falle von Apache auch nochmals auf ein paar Grundeinstellungen ein.

3.4.1 Apache mit SSL

Um SSL nutzen zu können muss die Konfigurationsdatei `httpd.conf` den Eintrag `LoadModule ssl_module modules/mod_ssl.so` aufweisen. Des Weiteren ist es notwendig, einen Server-Namen/IP zu vergeben (`ServerName 192.168.1.156:80`).
Die Alias-Anweisung ermöglicht es, auf Dokumente im Dateisystem außerhalb des mit DocumentRoot festgelegten Pfades zuzugreifen. Folgende Direktive in der `httpd.conf` leitet die Anfrage `http://localhost/test` auf den Pfad `/usr/local/apache2/test` um:

```
# eigener Alias (für alle virtual hosts sichtbar)
Alias /test "/usr/local/apache2/test"
<Directory "/usr/local/apache2/test">
```

```
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

In der Datei `/conf/ssl.conf` werden hier beispielhaft Virtual Hosts für SSL angegeben (dieses Listing enthält nicht alle, sondern nur relevante Zeilen):

```
Listen 443
SSLMutex default

NameVirtualHost 192.168.1.229:443

# Standard-Host, wenn Anfrage sonst nirgends passt
<VirtualHost 192.168.1.229:443>
   # General setup for the virtual host
   DocumentRoot e:/server/apache2/htdocs
   ServerName notebook1200
   ServerAdmin root@notebook1200
   ErrorLog logs/error_log
   TransferLog logs/access_log

   SSLEngine on
   SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
   SSLCertificateFile e:/server/apache2/conf/ssl.crt/server.crt
   SSLCertificateKeyFile e:/server/apache2/conf/ssl.key/server.key
</VirtualHost>

# localhost mit SSL
<VirtualHost 127.0.0.1:443>
   ServerName localhost
   DocumentRoot E:/server/apache2/htdocs
   ErrorLog logs/error_log
   TransferLog logs/access_log

   SSLEngine on
   SSLCertificateFile conf/ssl.crt/server.crt
   SSLCertificateKeyFile conf/ssl.key/server.key
   SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-
   response-1.0
   SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
</VirtualHost>

# www.hs.de mit SSL
<VirtualHost 192.168.1.229:443>
   ServerName www.hs.de
   DocumentRoot E:/server/apache2/htdocs/hs
   ErrorLog logs/error_log
   TransferLog logs/access_log

   SSLEngine on
   SSLCertificateFile conf/ssl.crt/server.crt
   SSLCertificateKeyFile conf/ssl.key/server.key
   SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-
   response-1.0
   SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
</VirtualHost>
```

Listing 3: Apache-Konfigurationsdatei ssl.conf (Auszug)

Mit Virtual Hosts kann man mehrere Domains auf eine IP abbilden (`httpd.conf`):

```
NameVirtualHost 192.168.1.229:80

# Standard-Host
<VirtualHost 192.168.1.229:80>
   ServerAdmin root@notebook1200
   DocumentRoot E:/server/apache2/htdocs
   ServerName notebook1200
   ErrorLog logs/notebook1200
</VirtualHost>

# www.hs.de ohne SSL
<VirtualHost 192.168.1.229:80>
   ServerName www.hs.de
   DocumentRoot E:/server/apache2/htdocs/hs
```

```
</VirtualHost>
```

Mit der NameVirtualHost-Anweisung wird die IP-Adresse angegeben, unter welcher der Server Anfragen für namensbasierte virtuelle Hosts entgegennimmt. Die Direktive ServerName bestimmt den Rechnernamen und –port, den der Server dazu verwendet, sich selbst zu identifizieren.

Unter Linux müssen der Konfigurationsdatei `httpd.conf` manuell noch die Zeilen `DirectoryIndex index.htm index.html index.php index.html.var` und `Listen 443` hinzugefügt werden. Wenn HTML-Seiten (-Verzeichnisse) außerhalb `/apache2/htdocs/` liegen und Rechteprobleme beim Aufruf der Seiten über den Browser auftreten sollten, dann auch noch `User apache` und `Group apache` eintragen.

3.4.2 Selbstsignierte Zertifikate

Zur theoretischen Ausführung zu selbstsignierten Server-Zertifikaten in Kapitel 2.7 soll hier nun die praktische Anleitung zum Erstellen selbiger folgen. Wenn nach dem Common Name gefragt ist, dann ist damit der Domainname gemeint (z.B. www.segelecke.de). Der Reihe nach sind folgende Befehle auf einem Linux-System mit installiertem OpenSSL auszuführen:

- `openssl req -new -out server.csr`
- `openssl rsa -in privkey.pem -out server.key`
- `openssl x509 -in server.csr -out server.crt -req -signkey server.key -days 365`
- `openssl x509 -in server.crt -out server.der.crt -outform DER`

Die generierten Dateien `server.crt` und `server.der.crt` im Verzeichnis `apache2/conf/ssl.crt` und die Datei `server.key` im Verzeichnis `apache2/conf/ssl.key` ablegen. Hinweis: Unter Umständen kann es einige Minuten dauern, bis das Zertifikat in seiner zeitlichen Gültigkeit ordnungsgemäß als gültig bewertet wird.

3.4.3 Tomcat

Das auf CD-ROM gebrannte PDF „Vorgehen Tomcat-Konfiguration“ enthält einige Beispielkonfigurationen zu Hello-World-Applikationen, umgesetzt mit den Techniken Servlet, JSP, Custom Tag und Struts.

Verzeichnisstruktur der Web-Applikation DiLog

- **`/webapps`**
  Enthält die verschiedenen Webapplikationen (völlig entkoppelt).
- **`/webapps/dilog`**
  Enthält eine konkrete Applikation mit ihren in Unterordner verteilten Daten.
- **`/webapps/dilog/conf`**
  Enthält verschiedene Konfigurationsdateien (hauptsächlich XML-Dateien) und DTDs (Document Type Definitions).

- **`/webapps/dilog/logs`**
  Enthält Logdateien der Applikation und des Chats.
- **`/webapps/dilog/temp`**
  Wird von DiLog zur Zwischenspeicherung von PDFs etc. benutzt.
- **`/webapps/dilog/web/applet`**
  Hier sind das signierte und unsignierte Chat-Applet samt JAR-Datei abgelegt.
- **`/webapps/dilog/web/download`**
  Zur Zeit ist hier ein Java-JRE (Version 1.3.1 für Windows) gespeichert. Es wird wegen des Browser-Java-Plugins zum direkten Download angeboten.
- **`/webapps/dilog/web/html`**
  Die Schriftarten der Applikation (CSS-Datei) und eine unbenutzte HTML-Datei, welche die Vorlage für das DiLog-Design darstellt.
- **`/webapps/dilog/web/img`**
  Ein Verzeichnis mit allen Bildern, die in DiLog angezeigt werden (außer Webcam-Bilder).
- **`/webapps/dilog/web/jsp`**
  Enthält alle verwendeten JSPs, unterteilt nach Modulen.
- **`/webapps/dilog/webcampics`**
  Speicherort für alle Webcam-Bilder.
- **`/webapps/dilog/webcamtemp`**
  Dieses Verzeichnis wird für den Upload der Webcam-Bilder temporär benutzt.
- **`/webapps/dilog/WEB-INF`**
  Enthält alle Tag-Library-Descriptor-Dateien für Taglibs und die Dateien `web.xml` (Servlets werden dem Java-Container bekannt gemacht), `struts-config.xml` (Konfigurationsdatei für die Struts-Abläufe) und `tiles-config.xml` (Konfigurationsdatei für Tiles).
  Hinweis: WEB-INF muss direkt unter dem mit dem Manager gemappten Pfad (`/dilog`) angesiedelt sein.
- **`/webapps/dilog/WEB-INF/backup`**
  Bei jeder Kompilierung des Quelltextes mit Ant wird vorher eine Sicherungskopie der zu ersetzenden Java-Dateien gemacht, sodass eine fehlerhafte Änderung des Codes durch das Backup rückgängig gemacht werden könnte.
- **`/webapps/dilog/WEB-INF/classes`**
  Class-Dateien in Package-Struktur und Ressource-Dateien im Hauptverzeichnis.
  Hinweis: Alle Klassen sollten in einem Package organisiert sein, da sonst unter bestimmten Umständen der Zugriff auf manche Ressource-Dateien nicht klappt.
- **`/webapps/dilog/WEB-INF/lib`**
  Enthält alle von der Applikation benötigten JAR-Archive.
- **`/webapps/dilog/WEB-INF/src`**
  Kann die Java-Dateien in derselben Package-Form wie in `/classes` enthalten.

Dabei ist `/webapps` ist das Unterverzeichnis von `/tomcat5`.

Weitere Einstellungen und Konfigurationsmöglichkeiten

Der mitgelieferte Tomcat-Manager (`http://localhost:8080/manager/html`) erkennt neue (ordnungsgemäße) Webapplikationen automatisch und zeigt diese in seiner Tabelle an. Wenn in der Datei `/dilog/WEB-INF/web.xml` unterhalb des Tags `<web-app>`

```
<display-name>Applikationsname</display-name>
<description>Ein beschreibender Hilfstext</description>
```

eingetragen ist, dann wird die Applikation mit dem Text des Tags `<display-name>` im Tomcat-Manager angezeigt.
DiLog liegt als WAR-Datei gepackt vor. Das macht eine Installation besonders einfach, denn diese Datei enthält alles, was die Webapplikation benötigt. Der Manager bietet eine einfache Möglichkeit um die WAR-Datei hochzuladen und dann automatisch zu installieren (Schaltfläche „WAR file to deploy“).

Hinweis zur Entwicklung: Geänderte JSPs werden automatisch neukompiliert und stehen der Applikation in veränderter Form zur Verfügung, geänderte (und manuell kompilierte) Servlets nicht. Es ist ein „Reload“ im Manager erforderlich. Dasselbe gilt für Änderungen von/in Ressource-Dateien.
Warnhinweis: Es sei nochmals darauf hingewiesen, dass ein einfacher Klick auf „Undeploy“ ausreicht, um eine Webanwendung sofort und dauerhaft zu löschen.[31]

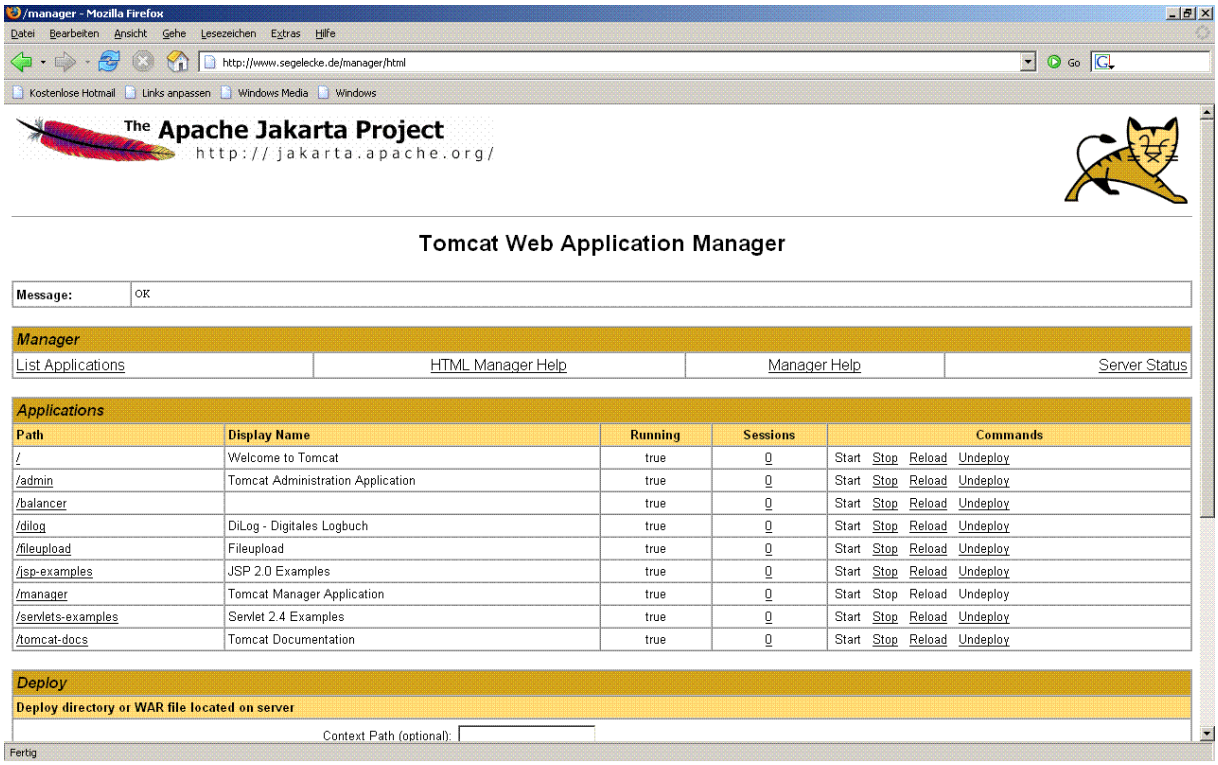


Abb. 5: Tomcat-Manager, Version 5

[31] Man beachte, wie gefährlich nahe der Link „Undeploy“ am häufig verwendeten Link „Reload“ platziert ist!

4 Software-Techniken im DiLog-Kontext

Dieses Kapitel beschreibt verschiedene Techniken, die DiLog direkt oder indirekt nutzt. Wo diese Benutzung unmittelbar sichtbar ist, soll auch darauf eingegangen werden, wie die jeweilige Komponente benutzt wird.

4.1 HTTP(S), Sessions, Cookies & Co.

> *Das World Wide Web basiert auf dem Hypertext Transfer Protocol HTTP. Client und Server verwenden ganz normale TCP-Verbindungen. Einige Spezialprotokolle erlauben neben dem allgemeinen HTTP-Zugang auch einen gesicherten Zugang über SSL oder TLS sowie den Zugriff auf und über Proxies, die z.B. Internet-Provider anbieten. Diese Spezialprotokolle verwenden eigene Ports.*

So beschreibt [Ziegler 2002] (S. 189) den Zusammenhang zwischen WWW und HTTP. Auf Verschlüsselung ging bereits Kapitel 2.7 ein, HTTPS macht sich SSL/TLS zunutze.
HTTP ist ein zustandsloses Datenaustausch-Protokoll der Anwendungsschicht im OSI-Modell und sitzt somit über TCP/IP. DiLog verwendet die Standardeinstellungen, sprich Port 80 für HTTP und (intern) Port 443 für HTTPS.

Ruft ein Browser beispielsweise die URL `http://www.segelecke.de/index.htm` auf um die HTML-Seite von einem entfernten Rechner (Server) auf den eigenen Rechner (Client) zu übertragen, so wird der Server `www.segelecke.de` per *HTTP-Request* aufgefordert, die Datei `index.htm` als *Response* zurückzusenden. Dazu sendet der Client über TCP eine GET-Anforderung auf Port 80 (`GET /index.htm HTTP/1.1`). Der Server antwortet im Erfolgsfall mit bestimmten Headern und der angeforderten Datei, welche der Browser dann darstellt. Ebenso können Bild-, Audio- oder Videodaten übertragen werden. Die Antwort könnte beispielsweise so aussehen:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1981
Server: Apache/2.0.49 (Unix) mod_ssl/2.0.49 OpenSSL/0.9.7d PHP/4.3.7 mod_jk2/2.0.2
Server at 147166.vserver.de Port 80
Connection: close
Last-Modified: Thu, 14 October 2004 01:03:37 GMT
```

Die erste Zeile enthält den Antwortstatus des Servers, die zweite den Inhaltstyp der übertragenen Datei (HTML), es folgt die Länge der Zeichen, Serverinformationen (zeigt alle installierten Apache-Module), Verbindungsstatus und Datum der Datei.

Bei Verwendung des älteren HTTP/1.0 wird bei jeder Anfrage eine TCP-Verbindung geöffnet und nach Übertragen der Antwort wieder geschlossen. Man kann sich leicht vorstellen, dass dies Ressourcen verschwendet, denn wenn eine HTML-Datei beispielsweise fünf Bilder enthält, so müssen insgesamt sechs Verbindungen geöffnet und geschlossen werden. Mit HTTP/1.1 kann eine Verbindung ohne sofortiges Schließen für mehrere Anfragen verwendet werden.

HTTP-Header können verschiedene Steuerinformationen wie Browsertyp, gewünschte Sprache, Anzahl der gesendeten Zeichen und Datentyp enthalten. Es ist auch möglich,

Header und Inhalt selbst zu definieren, also für eigene Anwendungen neue Header zu definieren. Header werden in Request-Header und Response-Header unterschieden, je nachdem ob sie einer Anfrage oder einer Antwort beigefügt sind. Die nachfolgende Tabelle aus [Turner 2003] (S. 71) listet einige der gebräuchlichsten Header auf. Dabei sind die ungeliebten 400er-Meldungen wohl die bekanntesten Antwortcodes.

Kategorie	***Antwortcode***	***Beschreibung***
1xx	100 Continue	Der Webserver hat den ersten Teil des Requests korrekt erhalten. Die Fortsetzung für den Rest folgt.
2xx	200 OK	Der Request wurde empfangen und korrekt verarbeitet. Die Response folgt.
3xx	301 Moved	Die angeforderte Ressource wurde verschoben. Die neue URI wird bereitgestellt, sodass der Browser eine neue Anforderung absetzen kann.
	304 Not Modified	Die angeforderte Ressource wurde nicht verändert. Die Seite kann sicher aus dem Cache geladen werden. Wird häufig für die Anforderung von Bildern verwendet.
4xx	401 Unauthorized	Für die angeforderte Ressource ist eine Berechtigung erforderlich. Der Benutzer sollte den Request mit den entsprechenden Berechtigungen erneut senden.
	403 Forbidden	Die Ressource steht dem Benutzer nicht zur Verfügung, unabhängig von der Authentifizierung.
	404 Not Found	Die angeforderte Ressource kann auf diesem Server nicht gefunden werden.
5xx	500 Server Error	Ein Fehler ist aufgetreten, während der Server versucht hat, den Request zu erledigen.

Abb. 6: HTTP-Antwortcode-Kategorien und Beispiel-Antwortcodes mit Beschreibungen

Ein Problem stellt die Zustandslosigkeit des Protokolls HTTP dar, denn Informationen der vorhergehenden Anforderung gehen bei der aktuellen Anfrage verloren. [Flanagan 1997] (S. 674) beschreibt sog. Cookies als allgemeinen " … Mechanismus, der von serverseitigen Verbindungen … verwendet werden kann, um Informationen auf der Client-Seite sowohl speichern als auch abrufen zu können.":

> *Wenn ein Server ein HTTP-Objekt an einen Client zurückgibt, kann er zusätzlich eine Zustandsinformation mitsenden, die der Client speichert. In diesem Zustandsobjekt ist auch eine Beschreibung des Bereichs von URLs enthalten, für die der Zustand gilt. Alle zukünftigen HTTP-Anfragen dieses Clients, die in den angegebenen Bereich fallen, übertragen den derzeitigen Wert des Zustandsobjekts zusammen mit der Anfrage zurück an den Server. Das Zustandsobjekt wird – ohne besonderen Grund – Cookie genannt.*

Um nun für die einzelnen Clients ein Session-Tracking zu ermöglichen, schreibt ein Servlet-Container eine zufällig generierte eindeutige Sitzungskennung in ein Cookie, das der Client-Browser von Seite zu Seite „mit sich herumschleppt". Anhand dieser Session-ID ist es dem Servlet-Container möglich, Benutzer A von Benutzer B zu unterscheiden, bzw. einen Benutzer wiederzuerkennen. Und nicht nur das: Ein Servlet kann beliebige Werte, Objekte und ganze Objektgeflechte in einer Session speichern, die nur für einen Client gelten (man spricht hier auch von Session-Beans). So sind z.B. Warenkörbe etc. aufgebaut, die sich eingekaufte Produkte (Objekte) merken. Andere Clients (Benutzer) haben darauf sinnvollerweise keinen Zugriff, bzw. haben eigene (evtl. auch gleiche) Objekte in ihrer

Session gespeichert. Cookies dürfen bis zu 4 KB groß sein und können beliebigen Text enthalten.

Was aber tun, wenn ein Benutzer in seinem Browser aus Furchtsamkeit den Cookie-Mechanismus ausschaltet?[32] Auch dafür gibt es eine Lösung: URL-Rewriting. Bei diesem Konzept wird jeder Link und jedes Formular, eben jede vom Anwender aktivierbare URL, vom Applicationserver umgeschrieben, bevor sie an den Client ausgehändigt wird. Dabei wird an jede URL ein sog. Session-Token angehängt. Dies entspricht der Session-ID, die in einem transienten Cookie gespeichert werden kann. Beim Login in DiLog kann man die Session-ID einmal als URL-Rewriting in der Adressleiste des Browsers erkennen (`https://www.segelecke.de/dilog/loadlogin.do;jsessionid=BE0B5B52AB2DCDBFC2E3B8F47CC3529B`).
URL-Rewriting ist offensichtlich weniger elegant, denn der Server benutzt es nur als letzten Ausweg, nämlich wenn Cookies nicht erlaubt sind.

Sessions (mit Hilfe von Cookies oder URL-Rewriting erreicht) speichern im digitalen Logbuch die Rolle eines Benutzers und dessen eindeutige, aus der Datenbank gelesene Benutzer-ID (Pers-ID). Der Session eines Administrators werden darüber hinaus einige System- und Anwendungswerte hinzugefügt.

4.2 Java Servlets

Java Servlets sind die Antwort auf den Ruf, mit Java-Technologie dynamische Webseiten zu ermöglichen. Im Gegensatz zu statischen Seiten eröffnen dynamische Seiten die Möglichkeit, sich oft ändernde Daten, Inhalte aus Datenbanken oder Daten, die von Benutzereingaben abhängig sind, im Browser anzuzeigen. Wie der Name schon andeutet, werden Servlets auf der Server-Seite ausgeführt, während die Client-Seite davon überhaupt nichts bemerkt (außer vielleicht einer JSessionId im Adressfeld des Browsers).

Servlets sind Java-Klassen, die laut J2EE[33]-Spezifikation direkt oder indirekt über andere abgeleitete Klassen von der Klasse `javax.servlet.HttpServlet` abgeleitet sind. Dabei werden die beiden Methoden `doGet()` und `doPost()` der Superklasse überschrieben[34]. Bei Bedarf erzeugt der Servlet-Container eine (weitere) Instanz der abgeleiteten Klasse. Die Methode `doGet()` dieses Objekts wird aufgerufen, wenn ein Client eine Anfrage an den Server richtet. `doPost()` kommt zum Einsatz, wenn der Client Formulardaten an den Server sendet. Somit reagiert der Server auf Client-Requests, indem er eine Verarbeitung anstößt und nach Verarbeitung (z.B. Datenbankzugriff) ein Ergebnis in den Ausgabestrom schreibt, der durch einen `PrintWriter` realisiert ist. HTTP-Request und -Response sind verfügbar

[32] Servlet-Container wie Tomcat verwenden nur Session-Beans, d.h. nachdem eine Session durch Ausloggen oder Schließen des Browser beendet wird, verfallen die Cookies. Es gibt auch dauerhaft auf der Festplatte des Benutzers gespeicherte Cookies, die z.B. dazu benutzt werden, einen Benutzer beim Login mit seinem Namen zu begrüßen. In Verruf geraten sind die Kekse deshalb, weil manche Anbieter solche über die Zeit angesammelten persistenten Cookies benutzen und (für den Benutzer unwissentlich) zu Marketingzwecken o.ä. auswerten, bzw. missbrauchen. So kann man auch Benutzerprofile über das Surfverhalten eines Menschen erstellen.

[33] Java 2 Enterprise Edition, siehe http://java.sun.com/j2ee/faq.html

[34] Es existieren weitere, aber unbedeutendere doXY()-Methoden.

über die Variablen `request` und `response`, welche beispielsweise die `doGet()`-Methode bereitstellt. `request` stellt alle Anfrageparameter und Formularwerte der Anfrage zur Verfügung, mit `response` kann man Response-Daten, Antwort-Header und die Antwortdaten an den Client übermitteln. Die Methode `getWriter()`, ausgeführt auf `response`, liefert den `PrintWriter`, in den nun mit einem normalen `println()` der statische HTML-Code, gemischt mit dynamischen Werten, geschrieben werden kann. Dabei wird ein Nachteil deutlich: Die Übersicht geht sehr schnell verloren, wenn der Code einer großen HTML-Seite in die `println()`-Methode gesteckt wird.

Der Lebenszyklus eines Servlets definiert sich durch die Abarbeitung dreier Methoden, die durch die Schnittstelle Servlet für Initialisierung, Abarbeitung der Anfragen und Beendigung vorgeschrieben werden: `init()`, `service()` und `destroy()`. Wie bereits angedeutet, stößt der Servlet-Container durch eine Client-Anfrage den ganzen Prozess an: Das Servlet wird durch einen Class Loader in den Speicher geladen, ein neuer Thread ruft die Methoden zur Abarbeitung auf. Die Methode `init()` wird nur einmal durchlaufen und kann z.B. eine Datenbankverbindung aufbauen oder sonstige initialen Arbeiten erledigen. Die Methode `service()` entstammt der Klasse `GenericServlet`, von der `HttpServlet` abgeleitet ist. `HttpServlet` implementiert die Methode `service()` praktischerweise so, dass Anfragen automatisch an die entsprechenden `doXY()`-Methoden weitergeleitet werden. Wird ein Servlet nicht mehr benötigt, so ruft der Container zum Ende `destroy()` auf, wobei hier die von `init()` bereitgestellten Ressourcen freigegeben werden oder sonstige finalen Aufräumarbeiten erledigt werden können.

```
package magtime.webcam.server;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
// weitere Imports

public class UploadServlet extends HttpServlet {

   static Logger logger = Logger.getLogger(UploadServlet.class.getName());

   private HttpServletResponse resp;
   private WebcamPics wp = new WebcamPics();
   private Hashtable parameters = new Hashtable();

   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws IOException, ServletException {

      logger.info("doGet()");

      resp = response;
      message("Dies ist ein Servlet zum FileUpload von Webcam-Bildern.\n(c) DiLog – Digitales
         Logbuch");

   }

   public void doPost(
      HttpServletRequest request,
      HttpServletResponse response)
      throws IOException, ServletException {

      logger.info("doPost()");

      resp = response;
      boolean isMultipart = FileUpload.isMultipartContent(request);
```

Listing 4: Webcam-Servlet zum Empfang von Bildern (verkürzte Darstellung)

```
        DiskFileUpload upload = new DiskFileUpload();

        // Parameter für Upload setzen
        upload.setSizeThreshold(500000); // memory 500 KB
        upload.setSizeMax(600000); // request 600 KB

        // weitere Codezeilen zur Verarbeitung der hochgeladenen Daten

        message("ok");

    }

    private void message(String msg) throws IOException {

        logger.info("message(" + msg + ")");

        resp.setContentType("text/plain");
        PrintWriter out = resp.getWriter();
        out.println(msg);

    }

}
```

Listing 4: Webcam-Servlet zum Empfang von Bildern (verkürzte Darstellung) (Fortsetzung)

4.3 JSP

Im vorhergehenden Unterkapitel wurde ein Nachteil von Servlets angesprochen: die Unübersichtlichkeit bei Ausgabe von viel HTML-Code. Dies ist die Stärke von Java Server Pages. Waren Servlets Java-Programme, die HTML-Code enthalten, so sind JSPs HTML-Dateien mit eingebettetem Java-Code. Intern werden JSPs allerdings vom Compiler in Servlets umgewandelt, doch das braucht den Entwickler nur insofern zu kümmern, als dass er dem Servlet-Container (bei DiLog Tomcat) ein J2SDK zur Verfügung stellt. Dies geschieht beim ersten Aufruf einer JSP und wiederholt sich nur dann, wenn die JSP verändert wurde.

JSPs können (neben gewöhnlichem HTML-Code) drei unterschiedliche Konstrukte enthalten. [Ullenboom 2003] erklärt sie ausführlich auf den Seiten 958 – 960 und 970 – 974. Hier soll ein sehr kurzer Überblick gegeben und die für DiLog wichtigsten Konstrukte skizziert werden:

- **Skript-Elemente** enthalten Programmcode, der direkt in das Servlet wandert. Dabei ist zu unterscheiden zwischen
  - Scriptlets, die beliebigen Java-Code einbetten können,
    `<% String managerlink = "http://" + request.getServerName() + ":8080/manager/html"; %>`
  - Ausdrücken, die eine Abkürzung für `out.println()` in Scriptlets darstellen
    `<html:link href="<%= managerlink %>" target="_blank">` und
  - Deklarationen, mit denen Objektvariablen und Methoden definiert werden können.
    `<%! int declaration = 1; %>`
- **Direktiven** steuern die Struktur der Seite und erzeugen keine sichtbare Ausgabe. Neben der include-Direktive zum Einbinden von gleichbleibenden HTML- oder JSP-Dateien gibt es die komplexere Page-Direktive mit den (ausgewählten) Attributen
  - contentType, das den MIME-Typ und den verwendeten Zeichensatz festlegt.
    `<%@ page contentType="text/html;charset=iso-8859-1" %>`

    - errorPage, das auf eine andere JSP verweist, die geladen wird, wenn durch einen Fehler eine Exception geworfen wird.
      `<%@ page errorPage="/web/jsp/errors.jsp" %>`
    - import, das wie bei herkömmlichen Java-Dateien eine oder mehrere Klassen importiert.
      `<%@ page import="magtime.struts.beans.BoatData" %>`
- **Aktionen** verändern das Verhalten des Servlet-Containers. Beispielsweise kann die Aktion
    - forward veranlassen, dass die Steuerung sofort und ohne Ausgabe auf eine andere JSP- oder HTML-Datei übergeht.[35]
      `<jsp:forward page='<%= pagename + ".htm" %>' />`
    - include wie die include-Direktive andere Dateien einbinden. Im Gegensatz zur Direktive `include` ist die Aktion include aber dynamisch, d.h. sie kann also zur Laufzeit auf unterschiedliche Seiten verweisen.
      `<jsp:include page="url" />`

Ein weiterer Vorteil ist, dass JSPs im Gegensatz zu Servlets von Webdesignern mit ihren gewohnten Werkzeugen wie Dreamweaver, GoLive etc. zu bearbeiten sind.
Es ist noch anzumerken, dass der Trend in Anlehnung an die strikte Trennung von Präsentation und Model/Controller beim MVC[36]-Entwurfsmuster dahin geht, möglichst wenig oder keinen Java-Code in JSPs einzubetten. Dasselbe Ziel verfolgt Struts mit seinem MVC2-Pattern (siehe Kapitel 4.4.2).

4.4 Das Framework Struts

Das Struts-Framework ist derzeit bei Version 1.2.4 angelangt. DiLog benutzt noch eine etwas ältere Version, nämlich 1.1b2. Dies ist u.a. darauf zurückzuführen, dass – einmal eingerichtet und funktionierend – im laufenden Projekt lieber nichts mehr daran geändert wurde.[37] Es gab durchaus irritierende Kuriositäten was Struts betrifft, doch prinzipiell ist Struts ausgereift und wird von einer Vielzahl von Java-Entwicklern auf der ganzen Welt benutzt und vorangetrieben (denn es ist wie fast alle in DiLog verwendeten Komponenten auch eine Open Source Software).

Ein Framework ist eine Kombination aus Software und Methoden, welche die Applikationsentwicklung beschleunigen und vereinfachen sollen. Jedoch empfiehlt die Struts-Homepage[38], bei einfacheren und kleinen Projekten Struts nicht zu verwenden. Der Overhead wäre in diesem Fall einfach zu groß. Stattdessen sollte man einen dem MVC(1)-Entwurfsmuster ähnlichen Ansatz wählen.
Für größere webbasierte Applikationen ist Struts jedoch ideal, weil es einen Ansatz und Einstieg bietet. Der Struts-Architektur liegt ein Model-View-Controller-Pattern zugrunde. Das Modell ist für die interne Darstellung der Daten verantwortlich und stellt die Verbindung zu Geschäftslogik und –prozessen dar. Die View zeigt zwar Daten an, ist aber, so gut es geht, von der Geschäftslogik entkoppelt. Der Controller steuert das Ganze, indem er festlegt,

35 Diese Aktion wird in DiLog durch das gleichwertige logic-Tag forward von Struts ersetzt.
36 Model-View-Controller
37 „Never touch a running system"! Dies gilt zumindest dann, wenn es nicht nur läuft, sondern gut läuft…

welche Schritte nacheinander ausgeführt werden sollen. Eine Sammlung von mehr oder weniger einfach zu verwendenden JSP-Tags (in Struts-Bibliotheken zusammengefasst) unterstützt die View, bietet zusätzliche Funktionalität, machen JSPs besser lesbar und einfacher wartbar. So funktionieren auch alle JSPs in DiLog durch Verwenden dieser Tag-Bibliotheken ziemlich ähnlich.

4.4.1 MVC-Pattern

> *Ein Entwurfsmuster besteht aus mehreren Objekten sowie Beziehungen zwischen diesen Objekten, die eine erprobte, erweiterbare Lösung für ein bestimmtes Problem des Softwareentwurfs darstellen. Das MVC-Muster (Model-View-Controller-Muster) ist nachweisbar das bekannteste und berühmteste Entwurfsmuster von allen.*

Das sind die einleitenden Worte von [Turner 2003] (S. 27) zum MVC-Muster, das Ende der 70er Jahre am Xerox Palo Alto Research Center entwickelt wurde. Es war ein Ansatz, um die GUI und die Benutzerinteraktion auf den ersten fensterbasierten Computern zu verwalten. Die Applikationen, um die es dabei ging, hatten – wie die meisten Anwendungen auch heute noch – eine Datenbankschicht, die Benutzerdaten o.ä. zu verwalten hatte, eine Darstellungsschicht, die dem Anwender die gewünschten Daten präsentieren musste, und eine Logik, die diese Abläufe steuerte. Die Logik war dafür verantwortlich, welche Anzeige zu sehen war, was im Fehlerfall passieren sollte und wann und wie das Backend aktualisiert werden sollte.

Äquivalent dazu benutzt Struts Model-Komponenten, welche Verbindung zur Geschäftslogik schlagend entfernte Systeme oder Datenbanken verwalten. Views werden mit JSPs realisiert und Controller-Komponenten, die im Prinzip nichts anderes tun als die Logik der ersten MVC-Applikationen, regeln und steuern alle Abläufe. MVC trennt die einzelnen Bestandteile voneinander, somit wird die Anzahl der Verknüpfungen untereinander minimiert. Entkopplung ist zudem eine wichtige Voraussetzung für wiederverwertbaren Code, was man mit Design Patterns ja schließlich auch erreichen will.

Doch noch war es nicht so weit: Am 7. Oktober 1998 veröffentlichte Sun die JSP-Spezifikation 0.92, die einen Abschnitt enthielt, der mit „JavaServer PagesAccess Model(s)“ bezeichnet war. Model 1 beschrieb die damals aktuelle Methode einer JSP-Verarbeitung: Der HTTP-Request wurde direkt an eine JSP geleitet, in der dann auch die gesamte weitere Verarbeitung stattfand.[39] Am Ende schickte dieselbe JSP das kalkulierte Ergebnis zurück an den Client (womöglich erfolgte die Darstellung wieder mit derselben JSP-Datei). Servlets schienen unpraktisch und wurden in den Hintergrund gedrängt.

4.4.2 MVC2-Pattern

Model 2 schrieb vor, dass die erste Anfrage eines Clients an ein Servlet gerichtet werden soll. Dieses arbeitet alle Aufgaben ab und speichert die Ergebnisse in einem Bean. Dieses Bean wurde an ein JSP übergeben, welches die Informationen aus dem Bean las, mit HTML-

[38] http://struts.apache.org

[39] So lernte ich Servlets und Java Server Pages im Jahr 2001 auch kennen.

Elementen kombinierte und zurück an den Client schickte. Jedoch wurde MVC nirgends in dieser Spezifikation genannt.

Über ein Jahr später machte Govind Seshadri mit einem Artikel darauf aufmerksam, dass Model 2 prinzipiell der MVC-Architektur entsprach. Im März 2000 startete Craig McClanahan, der zuvor am Tomcat-Projekt arbeitete, das Struts-Projekt als Teilprojekt von Apache-Jakarta. Später – mit zunehmendem Erfolg – wurde Struts ein eigenständiges Projekt der Apache Software Foundation.

Abbildung 7 zeigt die einzelnen Struts-Komponenten und ihre Interaktion im MVC-Muster als Kollaborationsdiagramm nach der Unified Modeling Language[40]. Abbildung 8 zeigt ein Sequenzdiagramm, das im Prinzip denselben Sachverhalt darstellt. Um es mit [Balzert 2001] zu sagen: „Während Sequenzdiagramme den zeitlichen Aspekt des dynamischen Verhaltens hervorheben, betonen Kollaborationsdiagramme die Verbindungen (*links*) zwischen Objekten.“.

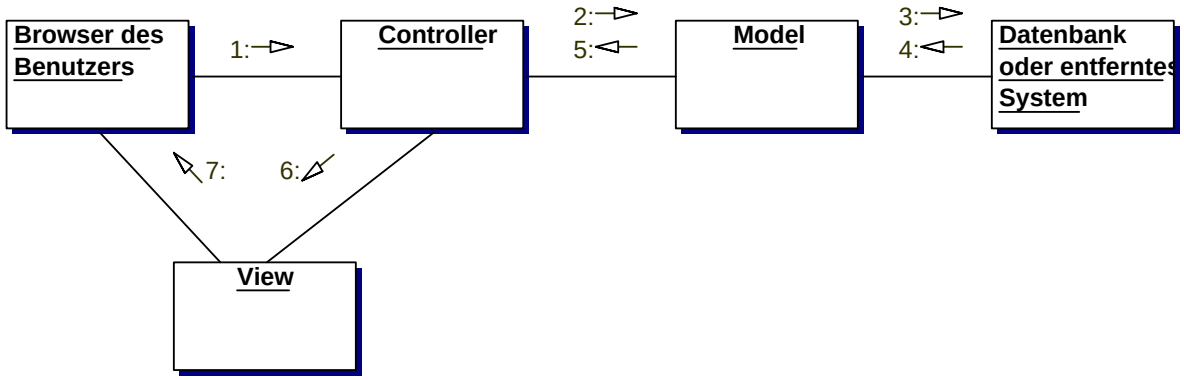


Abb. 7: Struts-Hauptkomponenten und ihre Interaktion nach dem MVC-Pattern

1. Der Client setzt einen Request an den Server ab.
2. Der Controller empfängt den Request und entscheidet abhängig von den in ihm implementierten Regeln und Anweisungen, wohin er die Steuerung abgibt (hier an das Model).
3. Das Model kapselt die Datenbank oder entfernte Systeme und holt sich von diesen Daten.
4. Dem Controller ist völlig egal, woher das Model die zu verarbeitenden Daten beschafft. Deshalb ist das Backend einfach austauschbar.
5. Das Model hat evtl. Daten verändert, die nun angezeigt werden müssen.
6. Der Controller entscheidet in Abhängigkeit der Verarbeitungsergebnisse, welches View-Objekt angezeigt werden soll. Daten werden im Controller zur Anzeige vorbereitet.
7. Daten werden in der View angezeigt, der Benutzer kann einen weiteren Zyklus anstoßen.

[40] 1994 taten sich Grady Booch und Jim Rumbaugh (Rational Software Corporation) zusammen und publizierten ein Jahr später erfolgreiche Methoden als Unified Method 0.8. Mit weiteren Partnern (Ivar Jacobson) und Firmen zusammen wurde UML 1.0 1997 der OMG (Object Management Group, http://www.omg.org) zur Standardisierung vorgelegt. Momentan ist UML-Version 1.5 aktuell (http://www.uml.org/#UML1.5).

4.4.3 Abläufe und Datenflüsse mit Hilfe von Java Beans

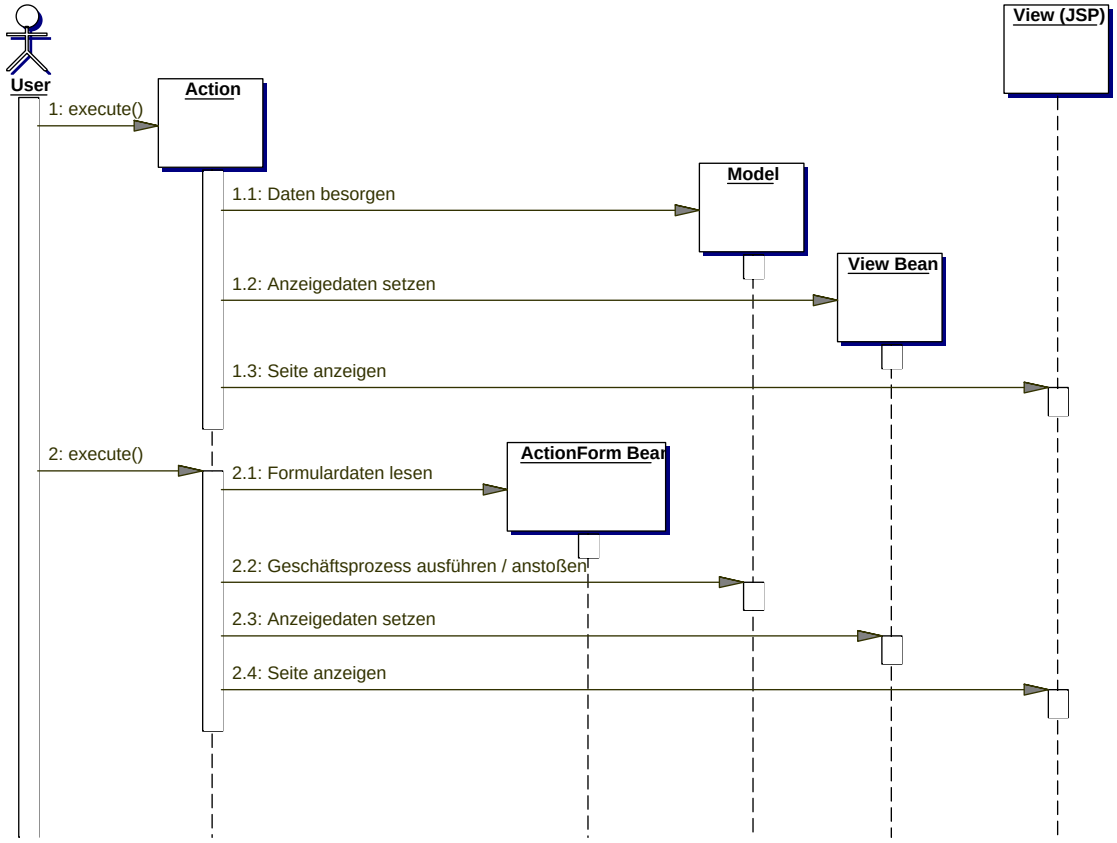


Abb. 8: Struts-Komponenten und ihre Interaktion nach dem MVC-Pattern

In Abbildung 8 werden auch die in Abbildung 7 nicht gezeigten Beans visualisiert. Dieses Sequenzdiagramm zeigt die Vorgänge etwas detaillierter und so, wie sie in DiLog häufig vorkommen. Es ist unschwer zu erkennen, dass der Benutzer zweimal agiert, also zweimal die Controller-Methode `execute()` der Action aufruft. Dieses Schema ist notwendig, wenn dem Benutzer zu allererst Daten aus der Geschäftslogik oder der Datenbank präsentiert werden müssen (Prozess 1.x). Dabei werden – wie MVC-Model 2 (oder einfach MVC2) das fordert – die aus dem Backend gelesenen Dateien in ein Bean gepackt und an die View (JSP) zur Anzeige geschickt. Dann gibt der Benutzer neue oder veränderte Daten in ein Formular ein, die Daten werden in einem ActionForm Bean gespeichert und vom Controller ausgelesen, nachdem dieser wieder aktiviert wurde (Prozess 2.1). Danach geht alles seinen bekannten Gang…

Teilweise trifft auf diese Beans zu, was auch auf die Session-Beans im EJB-Umfeld zutrifft ([Alur 2002] (S. 72)): Das Bean

- *ist für einen einzelnen Client oder Benutzer reserviert,*
- *besteht nur für die Dauer der Clientsitzung,*
- *übersteht keine Containerabstürze,*
- *ist kein persistentes Objekt,*

ActionForm Beans müssen für jede Variable, mit der sie einen Wert aus dem HTML-Formular speichern, eine `getX()`-Methode und eine `setXY()`-Methode besitzen (ganz normale Getter und Setter eben). Am einfachsten werden für alle Werte Strings benutzt, die im Controller dann konvertiert werden können. Für das folgende Formular-Beispiel (`/dilog/web/jsp/login/login.jsp`) muss das Bean so aussehen:

```
package magtime.struts.login;

// verschiedene Imports

public class LoginForm extends BaseActionForm {

   static Logger logger = Logger.getLogger(LoginForm.class.getName());
   private String identity = "";
   private String password = "";

   public void reset(ActionMapping mapping, HttpServletRequest request) {

      logger.info("reset()");
      this.identity = "";
      this.password = "";

   }

   public ActionErrors validate(
      ActionMapping mapping,
      HttpServletRequest request) {

      logger.info("validate()");

      ActionErrors errors = new ActionErrors();
      if (isEmpty(identity)) {
         errors.add(
            "login.kennung.leer",
            new ActionMessage("login.kennung.leer"));
      }
      if (isEmptySecure(password)) {
         errors.add(
            "login.passwort.leer",
            new ActionMessage("login.passwort.leer"));
      }

      return errors;

   }

   public String getIdentity() {
      return identity;
   }

   public String getPassword() {
      return password;
   }

   public void setIdentity(String string) {
      identity = string;
   }

   public void setPassword(String string) {
      password = string;
   }

}
```

Listing 5: ActionForm Bean LoginForm (verkürzte Darstellung)

Listing 5 zeigt die Klasse `magtime.struts.login.LoginForm` in etwas gekürzter Fassung. Neben den Import-Anweisungen wurden alle JavaDoc[41]-Kommentare entfernt um Platz zu sparen. Zu erkennen sind die beiden Instanzvariablen für Kennung und Passwort sowie die zugehörigen Getter und Setter. Fehlen diese, so quittiert das Framework beim Aufruf der

[41] Mit dem Dokumentationswerkzeug JavaDoc kann aus dem Quelltext heraus eine in sich verlinkte HTML-Software-Dokumentation generiert werden. Im Falle von DiLog kann dies ein Ant-Skript automatisieren.

korrespondierenden JSP-Datei mit dem Formular seinen Dienst mit einer Fehlermeldung. Des Weiteren kann man eine `validate()`-Methode entdecken; sie ist von der Basisklasse abgeleitet und kann für Datenvalidierung überschrieben werden. Die Prüfmethoden `isEmpty()` und `isEmptySecure()` stammen aus der Basisklasse `BaseActionForm`. Beide Methoden überprüfen ganz einfach, ob der Benutzer beim Login pro Textfeld mindestens ein Zeichen eingegeben hat. Falls eines der Textfelder tatsächlich leer sein sollte, wird eine `ActionMessage` erzeugt, die eine Fehlermeldung aufnimmt. Die Meldung kommt aus einem sog. Resource Bundle und wird über den Schlüssel `magtime.struts.login` gefunden. Wenn `validate()` kein leeres `ActionErrors`-Objekt zurückgibt, springt die Verarbeitung sofort wieder zurück zur View. Dort kann man sinnvollerweise die unter dem Schlüssel der Ressourcendatei hinterlegte Fehlermeldung anzeigen.

```
<html:form action="/login.do" focus="identity">

   <!-- Bean erstellen, wenn ein Fehler auftrat -->
   <html:messages bundle="login" id="msg">
      <bean:define id="hasErrors" value="true" />
   </html:messages>
   <html:messages bundle="login" id="msgController" message="true">
      <bean:define id="hasErrors" value="true" />
   </html:messages>

   <!-- wenn Bean = Fehler vorhanden -->
   <logic:present name="hasErrors">

      <table width="300" border="0" cellspacing="0" cellpadding="0">
         <tr>
            <td width="100"><span class="text">Kennung:</span></td>
            <td><html:text property="identity" size="20" maxlength="20" /></td>
         </tr>
         <tr>
            <td width="100"> </td>
            <td> </td>
         </tr>
         <tr>
            <td width="100"><span class="text">Passwort:</span></td>
            <td><html:password property="password" size="20" maxlength="20"
               redisplay="false" /></td>
         </tr>
         <tr>
            <td width="100"> </td>
            <td><html:submit property="submit" value="Anmelden" /></td>
         </tr>
      </table>

   </logic:present>

</html:form>
```

Listing 6: Auszug aus der Login-Datei login.jsp (Formularanzeige)

`<html:text property="identity" size="20" maxlength="20" />` zeichnet ein Textfeld mit dem Namen identity, in das man max. 20 Zeichen eingeben kann. Das zusätzliche Attribut `redisplay` legt fest, ob das Passwort nach Verlassen der Seite und Zurückkehren zur Seite (z.B. wegen falscher Dateneingabe) neu eingegeben werden muss oder nicht. `<html:submit property="submit" value="Anmelden" />` zeigt einen Button, beschriftet mit dem Wort „Anmelden“. Dieser Wert wird bei Betätigung der Schaltfläche auch übertragen (wenn eine Instanzvariable namens `submit` in der ActionForm vorhanden ist).
`<html:form action="/login.do">` verweist auf die in der Struts-Konfigurationsdatei definierte Action `login.do`, d.h. diese Action wird nach Betätigen des Submit-Buttons ausgeführt. Das leitet nahtlos zur Konfiguration von Struts über…

4.4.4 Struts-Konfiguration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
   "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>

   <global-exceptions>
      <!-- Fehlerseite für alle auftretenden Exceptions (außer JSPs) -->
      <exception  key="global.exception.allgemein" bundle="global"
         handler="magtime.struts.GlobalExceptionHandler"
         type="java.lang.Throwable" path="dilog.errorsfrombehind" />
   </global-exceptions>

   <global-forwards>
      <!-- Login von Admin gesperrt (hat keine Tiles-Schablone) -->
      <forward name="loginlocked" path="/web/jsp/loginsystemlock.jsp" />

      <!-- Alle Seitenzugriffe von Admin gesperrt (hat keine Tiles-Schablone) -->
      <forward name="totallocked" path="/web/jsp/totalsystemlock.jsp" />
   </global-forwards>

   <form-beans>
      <!-- FormBean für Loginformular -->
      <form-bean name="loginform" type="magtime.struts.login.LoginForm" />
   </form-beans>

   <action-mappings>
      <!-- Loginseite laden (entweder mit SSL oder SSL-Erklärungsseite anzeigen) -->
      <action path="/loadlogin" type="magtime.struts.login.TestSSLAction" >
         <forward name="sslok" path="dilog.login.login" />
      </action>

      <!-- Loginseite laden ohne SSL -->
      <action path="/loadloginwithoutssl" forward="dilog.login.login" />

      <!-- Loginformular -->
      <action path="/login" type="magtime.struts.login.LoginAction"
         name="loginform" scope="request" validate="true" input="dilog.login.login"
      >
         <forward name="loginok" path="dilog.start" />
         <forward name="loginbad" path="dilog.login.login" />
         <forward name="gotologininitial" path="dilog.login.badlogin" />
         <forward name="firstlogin" path="dilog.login.setperslogindata" />
      </action>

      <!-- Benutzers ausloggen -->
      <action path="/logout"
            type="magtime.struts.login.LogoutAction"
            scope="request"
      >
         <forward name="end" path="dilog.login.logout" />
      </action>
   </action-mappings>

   <message-resources parameter="login" key="login" null="false" />
   <!-- Scheinbar undokumentiert, aber vermeidet Exception "Cannot
       find message resources under key org.apache.struts.action.MESSAGE"
       beim Einsatz des Tags <bean:write> -->
   <message-resources parameter="resources.application"/>

</struts-config>
```

Listing 7: Auszug aus der Struts-Konfigurationsdatei struts-config.xml

Über die Datei `/dilog/WEB-INF/struts-config.xml` lässt sich der Controller steuern. Hier ist hinterlegt, welche Dateien, bzw. welche Tiles-Definitionen als nächstes wie verarbeitet werden sollen. Diese Datei ist sehr stark gekürzt, denn normalerweise hat sie über 820 Zeilen, d.h. sie ist nicht einmal zu einem Zehntel abgedruckt.

Ganz vorne im Dokument werden globale Exceptions behandelt. DiLog muss nicht zwischen unterschiedlichen Exceptions unterscheiden, sondern kann alle Exceptions vom Typ

Throwable an den Handler `magtime.struts.GlobalExceptionHandler` weitergeben. Dieser speichert die Exception-Daten und leitet sie zur Ausgabe weiter. Es schließen sich globale Forwards an, die keine Tiles-Schablonen benutzen und deshalb im Path-Attribut direkt auf eine JSP verweisen (so würde das gemacht, wenn man nicht Tiles verwenden würde). Mit dem Tag `<form-bean>` werden die ActionForm Beans definiert. Dabei gibt der Type selbstverständlich die Klasse an, die als Formular instantiiert werden soll.
Es folgt der umfangreichste Teil, die Action-Mappings. Wenn man das Login-Formular betrachtet, so fällt einem der Wert des Attributs `path` auf (`/login`); es ist derselbe wie in der JSP-Datei mit dem Formular, nur ohne „.do“ (Listing 6, erste Zeile). `type` gibt an, welche Action (Teil des Controllers) ausgeführt werden soll. `name` setzt das zuvor oben schon definierte ActionForm Bean, `scope` gibt die Gültigkeitsdauer an, `validate`, ob in dem Formular-Bean eine Prüfung durchgeführt werden soll, und `input` gibt an, woher die Seite kommt. Im Fehlerfall muss die Steuerung an diese Seite zurückgegeben werden. Es folgen beliebig viele Weiterleitungen mit Key/Value-Paaren. Die Schlüssel können innerhalb der Action mit dem Konstrukt `return mapping.findForward("key");` angegeben werden.
Eine Action kann auch wie `/loadloginwithoutssl` nur mit den Attributen `path` und `forward` angegeben werden. Es gibt hierbei sehr viele mögliche Kombinationen. Abschließend folgt hier noch die Definition von Message-Ressourcen, die Dateien mit Key/Value-Inhalt müssen im Klassenpfad liegen. Um Struts dem Servlet-Container überhaupt bekannt zu machen, sind Einträge in der Datei `web.xml` nötig (siehe Anhang).

4.4.5 Schablonen mit Tiles

Action-Definitionen in der Datei `struts-config.xml` geben als Pfad einer Weiterleitung fast immer eine Tiles-Definition an. Tiles ist noch nicht so lange Bestandteil von Struts, dementsprechend knirschte es ab und an mit der von DiLog verwendeten Struts-Version. So kann das Taglib `<logic:forward>` beispielsweise nicht auf Tiles-Definitionen weiterleiten. Es muss immer der Umweg über eine zwar kleine, aber zusätzliche Action gegangen werden.

Unabhängig von solchen Unwegbarkeiten ist es aber dennoch nützlich, Tiles zu verwenden. Viele (dynamische) Websites haben neben wirklich dynamischen Teilen auch mehr oder weniger statische Anteile wie z.B. Header, Footer oder ein gleichbleibendes Navigationsmenü. Die Tiles-Taglib macht es möglich, eine Seitenschablone – definiert in XML – für die gesamte Website zu verwenden. Dabei arbeitet Tiles so eng mit Struts zusammen, dass, wie schon gesehen, frei wählbare Layout-Definitionen anstelle von JSP-Dateinamen in der Struts-Konfigurationsdatei verwendet werden können.

Wie dies mit Schablonentechnik üblich ist, kann man mit Veränderung einer Layoutdatei das Aussehen der gesamten Website auf einen Schlag bequem ändern. Ein weiterer Vorteil besteht darin, dass die Inhaltsseiten nichts über ihre Umgebung wissen müssen, sodass derselbe Inhalt in mehreren verschiedenen Websites verwendet werden könnte. Manche Entwickler sehen es als zusätzlichen Vorteil an, dass die JSP-Dateinamen mit Verwendung von Tiles aus `struts-config.xml` verschwinden. Darüber kann man sich allerdings streiten, da die logischen Dateinamen nun eben in einer anderen Datei vorgehalten werden. Man könnte es sogar als kleinen Nachteil werten, nun in zwei Dateien blättern zu müssen…

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
  "-//Apache Software Foundation//DTD Tiles Configuration//EN"
  "http://jakarta.apache.org/struts/dtds/tiles-config.dtd">

<tiles-definitions>

   # ------------------- Schablonen ---------------------------------------

   <!-- Standard-Schablone ohne Autorisierungsprüfung -->
   <definition name="dilog.default.unsecure" path="/web/jsp/dilog_unsecure.jsp">
      <put name="header" value="/web/jsp/header.jsp" />
      <put name="navigation" value="/web/jsp/navigation.jsp" />
   </definition>

   <!-- Standard-Schablone mit Autorisierungsprüfung -->
   <definition name="dilog.default.secure" path="/web/jsp/dilog_secure.jsp">
      <put name="header" value="/web/jsp/header.jsp" />
      <put name="navigation" value="/web/jsp/navigation.jsp" />
   </definition>

   # ------------------- Schablonen ---------------------------------------

   <definition name="dilog.errorsfrombehind" extends="dilog.default.unsecure">
      <put name="main" value="/web/jsp/errors.jsp" />
      <put name="navigation" value="/web/jsp/navigation_empty.jsp" />
   </definition>

   <definition name="dilog.start" extends="dilog.default.secure">
      <put name="main" value="/web/jsp/start.jsp" />
   </definition>

   <definition name="dilog.login.login" extends="dilog.default.unsecure">
      <put name="main" value="/web/jsp/login/login.jsp" />
      <put name="navigation" value="/web/jsp/navigation_empty.jsp" />
   </definition>

   <definition name="dilog.login.badlogin" extends="dilog.default.unsecure">
      <put name="main" value="/web/jsp/login/badlogin.jsp" />
      <put name="navigation" value="/web/jsp/navigation_empty.jsp" />
   </definition>

   <definition name="dilog.login.setperslogindata" extends="dilog.default.secure">
      <put name="main" value="/web/jsp/login/setperslogindata.jsp" />
      <put name="navigation" value="/web/jsp/navigation_empty.jsp" />
   </definition>

   <definition name="dilog.login.logout" extends="dilog.default.unsecure">
      <put name="main" value="/web/jsp/login/logout.jsp" />
      <put name="navigation" value="/web/jsp/navigation_empty_bottom.jsp" />
   </definition>

</tiles-definitions>
```

Listing 8: Auszug aus der Tiles-Konfigurationsdatei tiles-config.xml

Auch diese Datei ist stark verkürzt dargestellt (umfasst in DiLog über 400 Zeilen). Der Einfachheit halber liegen `struts-config.xml`, `tiles-config.xml` und `web.xml` in einem Verzeichnis und erstere wurden auch nicht in weiter in Module unterteilt, was theoretisch auch noch möglich wäre.
Hier sind alle Tiles-Definitionen aufgeführt, auf welche die Actions `login.do` und `logout.do` mit ihren Weiterleitungen in Listing 7 verweisen.

Um Tiles überhaupt benutzen zu können, muss folgendes in die `struts-config.xml` eingetragen und den persönlichen Anforderungen nach angepasst werden:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
   <set-property property="definitions-config" value="/WEB-INF/tiles-config.xml" />
   <set-property property="definitions-debug" value="2" />
   <set-property property="definitions-parser-details" value="2" />
   <set-property property="definitions-parser-validate" value="true" />
</plug-in>
```

Zusätzlich muss in die Datei `web.xml` eingetragen werden:

```
<taglib>
   <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
```

Sehen wir uns zunächst die Definition der Standardschablone `dilog.default.unsecure` an (Listing 8). Sie gibt als Pfad die Datei `/dilog/web/jsp/dilog_unsecure.jsp` an, welche Listing 9 zeigt:

```
<%@ page contentType="text/html;charset=iso-8859-1" language="java" %>
<%@ page errorPage="/web/jsp/errors.jsp" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>

<tiles:insert attribute="header" />
<tiles:insert attribute="navigation" />
<tiles:insert attribute="main" />
     </td>
  </tr>
</table>
<br>
<br>
</body>
</html>
```

Listing 9: Vorlagendatei dilog_unsecure.jsp

Hier kann man deutlich die drei Tiles-Tags erkennen, gefolgt von abschließendem HTML-Code. Anstelle dieser Tiles-Tags bindet Tiles mit Hilfe des Tags `put` automatisch die in der jeweiligen Tiles-Definition angegebenen JSP-Dateien ein, hier also die Dateien `header.jsp` und `navigation.jsp`. Die „unsichere“ und „sichere“ Standardschablone unterscheiden sich nur dadurch, dass die sichere Schablone bei jedem Seitenaufruf überprüft, ob die Session des Benutzers ein Attribut `persId` enthält. Dieses Attribut wird der Session bei der Anmeldung über die Login-Action hinzugefügt und ist für die ganze Benutzersitzung gültig. Das verhindert, dass ein potenzieller Angreifer durch Erraten von Dateinamen Seiten ohne vorherige Anmeldung (Authentisierung und Autorisierung) aufrufen kann. Ferner überprüft `dilog_secure.jsp` auch, ob der Administrator das System gesperrt hat. Wenn dies der Fall ist, erfolgt mit dem bereits erwähnten Struts-Logic-Tag `<logic:forward name="totallocked" />` über die globale Weiterleitung `totallocked` (Listing 7) eine sofortige Weiterleitung auf eine entsprechende Hinweisseite. Demzufolge wird die „sichere“ Standardschablone auch generell nach erfolgreichem Login benutzt. Dabei ist allerdings zu beachten, dass die Standardschablonen niemals direkt verwendet werden!

Wenn man die Tiles-Definition `dilog.login.login` in Listing 8 betrachtet, so fällt auf, dass sie von `dilog.default.unsecure` abgeleitet ist (Vererbungsprinzip). Sie übernimmt also die von der Standardschablone definierten Inhalte für `header` und `navigation`., kann die Werte aber auch überschreiben um eigene Inhalte zu definieren, was hier auch passiert um das Navigationsmenü auszublenden. Nun wird auch klar, warum `dilog.default.unsecure` das Tiles-Insert-Attribut `main` nicht mit `put` angibt; das tun nur die erbenden Tiles-Definitionen, da der mit `main` dargestellte Inhalt auf keiner Seite derselbe ist. Die ebenfalls in Listing 8 dargestellte Tiles-Definition `dilog.login.logout` verwendet für das Navigationsmenü nochmals eine andere Datei, was auf Design-Gründe zurückzuführen ist.

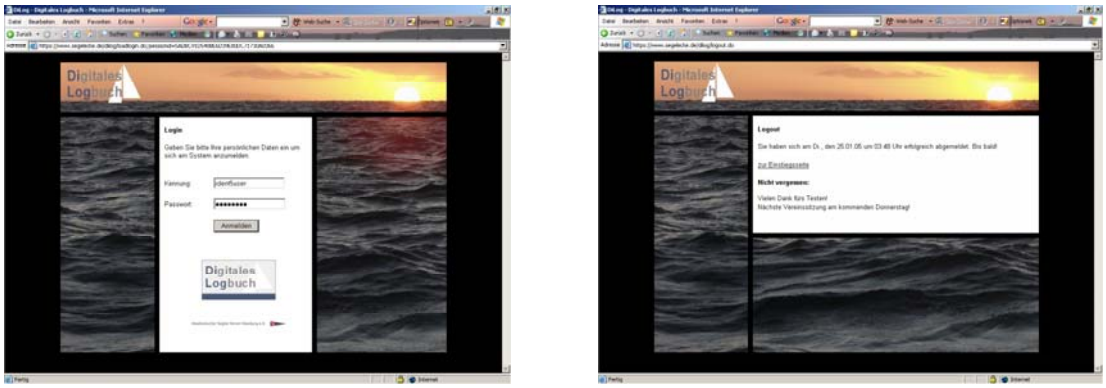


Abb. 9: Browser-Ansicht DiLog, Login und Logout

Um die Aufteilung der Weboberfläche in Segmente durch Tiles-Definitionen noch deutlicher zu machen, hier zwei weitere exemplarische Ansichten (Gast-Ansicht und Benutzer-Ansicht):

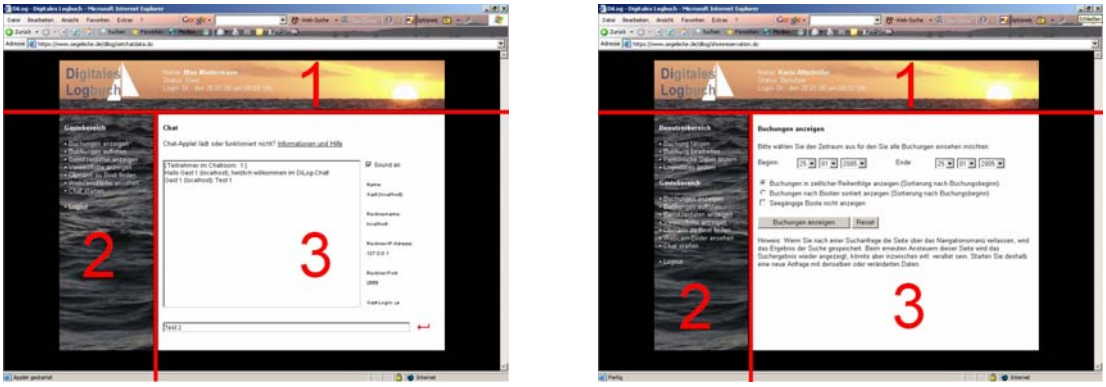


Abb. 10: Browser-Ansicht DiLog in Segmente unterteilt (Status Gast und Benutzer)

- Der Header-Bereich (1) wird – obwohl natürlich möglich – in der ganzen Applikation nicht durch eine von der Standardschablone abgeleitete Schablone (Tiles-Definition) ersetzt. Es gab einfach keinen Grund, die Anzeige zu ändern.
- Der Navigationsbereich (2) kennt einmal die normale und fast immer benutzte Darstellung des Navigationsmenüs, zum andern die seltenere Ausblendung des Menüs. Letzteres geschieht hauptsächlich dann, wenn der Benutzer gezwungen werden soll, die angezeigte Seite über einen bestimmten Link zu verlassen um z.B. mit Betätigen des Links eine zuvor generierte Datei vom Server zu löschen.
- Der Bereich zur Darstellung des ständig wechselnden Inhalts (3).

Es könnte jetzt der Eindruck entstehen, dass man nun auf dieses eine Schema festgelegt ist. Dem ist natürlich nicht so: Indem man auf die Ableitung verzichtet, kann man völlig unabhängige Designs erstellen. So öffnet beispielsweise der Eintrag `<definition name="dilog.guest.listreservationdetail" path="/web/jsp/guest/listreservationdetail.jsp" />` in der Tiles-Konfigurationsdatei ein Popup-Fenster um Details zu einer angeklickten Buchung zu zeigen.

Das Navigationsmenü selbst ändert seine Darstellung abhängig von der Systemrolle des eingeloggten Benutzers und der vom Benutzer gewählten Ansicht. Wenn das Menü geladen wird, fragt die JSP zuerst den Status des Benutzers ab (Admin, Obmann, Benutzer, Gast).

Direkt nach dem Login wird die Standardansicht gezeigt. Hierarchisch aufeinander aufbauend darf der Benutzer die Bereiche für Benutzer und Gast sehen, der Obmann die Bereiche Obmann, Benutzer und Gast (usw.). Ein anderer Bereich kann durch Betätigen eines Links aufgeklappt werden, dabei klappt der vorherige im Normalfall zu, was die Übersichtlichkeit erhöht (siehe Abb. 11). Der Action, die das Wechseln der Ansicht steuert, wird mit einem dem Link angehängten Parameter `view` die gewünschte Ansicht übermittelt. Die Action (`navlinkchangenavigationview.do`) liest den Parameter aus und aktualisiert das Attribut `chosenview` in der Session. Listing 10 zeigt diese (sehr einfache) Action, komplizierte Actions funktionieren jedoch nach demselben Prinzip: Es muss die geerbte Methode `execute()` implementiert werden. Um zu entscheiden, welche Ansicht geladen werden soll, liest `navigation.jsp` dieses Session-Attribut bei jedem Aufruf der Seite aus.

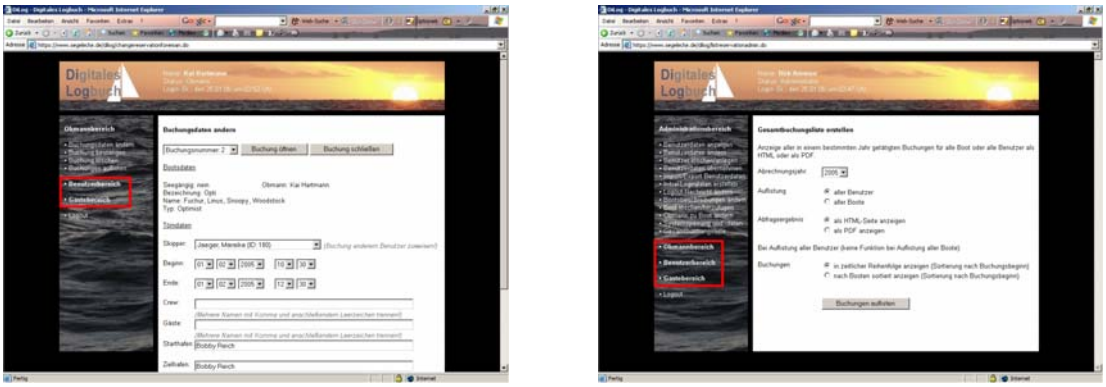


Abb. 11: Browser-Ansicht DiLog, Ansichten Obmann und Administrator

```
package magtime.struts;

// verschiedene Importanweisungen

public class ChangeNavigationViewAction extends Action implements SystemRoles {

   static Logger logger =
      Logger.getLogger(ChangeNavigationViewAction.class.getName());

   /**
    * Erzeugt ein Bean, das die gewünschte Ansicht enthält.
    */
   public ActionForward execute(
      ActionMapping mapping,
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response)
      throws Exception {

      logger.info("execute()");

      String navView = (String) request.getParameter("view");
      request.getSession().setAttribute("chosenview", navView);

      return mapping.findForward("changed");

   }

}
```

Listing 10: Beispiel einer Action: ChangeNavigationViewAction

4.4.6 Struts-Taglibs (Custom Tags)

Die Tag-Libraries von Struts sind im Prinzip nichts weiter als eine zusammengesammelte Menge an selbstdefinierten JSP-Tags (siehe 4.4.7), welche die JSP-Funktionalität aber nicht

unerheblich erweitern. Trotzdem vereinfachen sie gleichzeitig die Erstellung von JSPs, indem sie es möglich machen, Java-Scriptlets weitgehend zu vermeiden. Der Java-Code wird stattdessen in benutzerdefinierte Tags eingebettet. Zudem ist ein Tag oft leichter verständlich, zumindest für den Grafik-Designer einer Website. Die mit Struts mitgelieferten Tag-Libraries heißen Bean, HTML, Logic, Nested und Tiles und enthalten jeweils eine kleinere oder größere Sammlung von einzelnen Tags.

Tag-Libraries werden in eine JSP mit der Anweisung `<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>` eingebunden. Die URI verweist auf einen Tag Library Descriptor (TLD), der im XML-Format die Verbindung des einzelnen Tags zu jeweils einer serverseitig ausgeführten Java-Klasse herstellt. Das Präfix ist frei wählbar, unter diesem Namen sind die einzelnen Tags einer Taglib ansprechbar.

Listing 6 verwendete bereits einige Struts-Tags[42]. `<html:form action="/login.do" focus="identity">` ist das eröffnende Tag um ein Formular zu erstellen, dessen Daten von der Action `login.do` verarbeitet werden sollen. Dabei soll das Textfeld `identity` zu Beginn den Fokus erhalten.

```
<html:messages bundle="login" id="msg">
   <bean:define id="hasErrors" value="true" />
</html:messages>
```

Dieser Abschnitt überprüft, ob eine ActionForm oder eine Action (Zusatz `message="true"`) einen Fehler aus dem Resource Bundle login hinterlegt hat. Falls dies der Fall sein sollte, wird der verschachtelte Inhalt („nested body content“) angezeigt oder ausgeführt. Hier würde ein neues Bean mit dem Namen hasErrors und dem Wert true erstellt.
`<logic:present name="hasErrors">` untersucht, ob ein Bean mit dem Namen hasErrors vorhanden ist. Auch hier wird der zwischen dem öffnenden und schließenden Tag befindliche Inhalt nur angezeigt, wenn das Bean vorhanden ist. Es lassen sich also verschiedene Prüfungen vornehmen, und JSP-Code kann dadurch bedingt ausgeführt werden. Der Grund für die Verwendung in Listing 6 besteht darin, dass die beiden Bilder auf der Login-Seite nicht angezeigt werden sollen, wenn nach einem misslungenen Login-Versuch Fehlermeldungen angezeigt werden müssen.

Damit wären an dieser Stelle auch schon aus jeder Bibliothek ein oder mehrere Tags erläutert worden, was genügen soll. Auf der Struts-Homepage sind unter der Rubrik Developer Guides zu jeder Taglib und allen enthaltenen Tags ausführliche Informationen zu finden.

4.4.7 Eigene Taglibs

DiLog verwendet zwei selbstdefinierte Tags in der Taglib `memberwrapper`. Das `Person`-Tag ermöglicht die direkte Anzeige von persönlichen Daten eines Mitglieds wie etwa Name, Adresse und Telefonnummern, das `Member`-Tag kann die Mitgliedsdaten wie Segelberechtigungen, Verwaltungsinformationen und Eintrittsdatum eines Benutzers

[42] Es sei an dieser Stelle darauf hingewiesen, dass alle Tags XML-konform entweder mit einem öffnenden (<tagname>) und schließenden (</tagname>) Tag oder – wenn kein „tag body“ vorhanden ist – einfach mit einem Slash am Ende des Tags (<tagname />) verwendet werden müssen, auch wenn dies in den angeführten Beispielen nicht extra erwähnt ist.

anzeigen. Der TLD im Verzeichnis `/dilog/WEB-INF` definiert jeweils eine Java-Klasse und einige Attribute, darunter die Attribute `persId` und `property`, welche für beide Tags zwingend angegeben werden müssen. Der komplette TLD ist im Anhang abgedruckt.

Es wird nun der komplette Vorgang bei Verwendung des Tags `Person` simuliert. Zu Beginn der JSP `/dilog/web/jsp/start.jsp` wird der Speicherort des TLDs mit `<%@ taglib uri="/WEB-INF/memberwrapper.tld" prefix="mw" %>` bekannt gemacht. In dieser Datei ist definiert, dass die Klasse `magtime.struts.taglibs.memberwrapper.PersonTag` für den Aufruf `<mw:person persId="<%= persId %>" property="surname" />` zuständig sein soll. Der Ausdruck `<%= persId %>` als Abkürzung für ein Scriptlet fügt die zuvor aus der Session ausgelesene Benutzer-ID ein. Um das zu erlauben muss das Attribut `persId` im TLD mit dem Zusatz `<rtexprvalue>true</rtexprvalue>` versehen werden.

Nehmen wir an, Benutzer Scholz mit der ID 85 ruft die Seite `start.jsp` auf. Über das Tag und den Hinweis im TLD wird die Methode `doStartTag()` der Klasse `PersonTag` aufgerufen. Über Getter und Setter werden der Methode die PersId 85 und die gesuchte Eigenschaft surname übermittelt (wie das im Einzelnen geschieht, soll hier nicht weiter von Interesse sein). `doStartTag()` lädt nun den Benutzer mit der Benutzer-ID 85, stellt fest, dass der Nachname gesucht wird und ermittelt diesen. Der gefundene Name Scholz wird mit dem Aufruf `ResponseUtils.write(pageContext, out);` zur Ausgabe im Browser gebracht. Die Methode endet mit `return (SKIP_BODY);` was bedeutet, dass das Tag keinen Inhalt hat, die Verarbeitung hiermit also beendet ist. Zur Deutung der weiteren Attribute wird auf die der CD-ROM beigefügten JavaDoc-Dokumentation verwiesen.

4.4.8 Tipps zu Struts

Die im Folgenden geschilderten Probleme und Einschränkungen sind inzwischen mit einer neueren Struts-Version evtl. behoben. Jedoch kann es für den Entwickler, der sich in DiLog einarbeiten muss, sinnvoll sein, zu ergründen, warum ein bestimmtes Problem gerade so gelöst wurde.

Ablösung von ActionError durch ActionMessage

In Struts 1.1 wurde die Klasse ActionError als „deprecated“ markiert, mit dem Hinweis, stattdessen ActionMessage zu verwenden, weil die Klasse in Version 1.2 entfernt werden wird. Auf der Servlet-Seite funktionierte die Umstellung ohne große Änderungen (Listing 5 zeigt die Verwendung von ActionMessage), auf Seite der JSPs musste mehr geändert werden.

Mit dem einfachen Tag `<html:errors bundle="login" />` konnten alle vorhandenen Fehler angezeigt werden. Zudem konnte man mit den Eintragungen `errors.header=<font color="ff0000">`, `errors.footer=</font>` und `errors.suffix=<br>` in der Properties-Datei passabel die Formatierung beeinflussen. Damit werden die einzelnen Fehlertexte mit Zeilenumbruch und in Schriftfarbe rot ausgegeben. Dabei ist nur die Properties-Datei in der Datei `struts-config.xml` anzugeben: `<message-resources parameter="login"`

`key="login" null="false" />`. Nun müssen die Fehlermeldungen über das bereits beleuchtete (kompliziertere) Tag `<html:messages>` ausgelesen werden.

Die Fehlermeldungen werden in einem Java-Bean gespeichert und über das Bean-Tag `<bean:write name="msg" filter="false" />` ausgelesen. Dabei muss als `name` der über das HTML-Tag als `id` angegebene Name übernommen werden. Da – zumindest zur Zeit – keine Formatierungsangaben in der Properties-Datei hinterlegt werden können, müssen z.B. Zeilenvorschübe (`<br>`) direkt im Text der Fehlermeldungen angegeben werden. Der Wert für `filter` muss dabei auf false gesetzt werden, dann werden die in der Properties-Datei hinterlegten Texte (mit HTML-Tags) nicht automatisch HTML-codiert und somit als HTML-Tags erkannt. Allerdings müssen die Texte dabei schon HTML-codiert abgelegt werden.

Einschränkungen bei Global-Forwards und Tiles

In Struts kann man innerhalb der `execute()`-Methode einer Action als Rückgabewert für `ActionForward()` mit `return mapping.findForward("tilesDefName");` auf eine globale Weiterleitung verweisen. Das funktioniert so weit auch. Leider unterstützt das Tag `<logic:forward name="tilesDefName" />` in der verwendeten (inzwischen veralteten) Version keine Weiterleitung auf globale Weiterleitungen. „Keine Weiterleitung auf globale Weiterleitungen“ – man merkt an dieser Wendung im Prinzip schon, dass dies eine sehr unschöne Einschränkung darstellt.

Wenn man es trotzdem versucht, wird, anstatt in den Tiles-Definitionen nachzuschlagen, direkt auf `/servletContext/tilesDefName` verwiesen, was natürlich zu HTTP-Fehler 404 (Resource missing) führt. Der ServletContext lautet im Falle des digitalen Logbuchs `dilog`. Als Workaround kann entweder auf Tiles verzichtet und direkt auf eine JSP weitergeleitet werden (unschön)[43] oder auf eine Action weitergeleitet werden, die dann eine Tiles-Schablone aufruft (unpraktisch, da Overhead entsteht).

Problem mit File-Uploads

Solange man die mit Struts mitgelieferte Bibliothek `commons-fileupload.jar` verwendet, klappt alles bestens. Um den Upload mit dem Commons-HttpClient vornehmen zu können[44], muss die Struts unbekannte Bibliothek `commons-fileupload-1.0.jar` in den Klassenpfad aufgenommen werden. Dann klappt der Struts-Upload über ein HTML-Formular jedoch nicht mehr, da die neue Klasse `FileUpload` bzw. deren Superklasse `FileUploadBase` nicht mit den mit Struts mitgelieferten Klassen identisch ist. Sobald die 1.0-JAR-Datei verwendet wird, wirft das Servlet folgende Exception:

```
java.lang.NoSuchMethodError: org.apache.commons.fileupload.FileUpload.setSizeMax(I)V
org.apache.struts.upload.CommonsMultipartRequestHandler.handleRequest(
   CommonsMultipartRequestHandler.java:220)
org.apache.struts.util.RequestUtils.populate(RequestUtils.java:934)
org.apache.struts.action.RequestProcessor.processPopulate(RequestProcessor.java:779)
org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:246)
```

[43] Die JSP muss dann komplett mit Header und Navigation vorliegen.

[44] Dies wurde notwendig, da bestimmte Attribute im Request des HttpClients über Struts nicht erreichbar waren oder von Struts entfernt wurden (betrifft den Webcam-Server).

```
org.apache.struts.action.ActionServlet.process(ActionServlet.java:1292)
org.apache.struts.action.ActionServlet.doPost(ActionServlet.java:510)
javax.servlet.http.HttpServlet.service(HttpServlet.java:763)
javax.servlet.http.HttpServlet.service(HttpServlet.java:856)
```

Es blieb nichts anderes übrig, als die Package-Struktur des zum Glück als Source Code vorhandenen Commons Fileupload zu ändern (ein weiterer Grund, der für die Verwendung von Open Source Software spricht).

4.5 Java Applets

Begriffliches: Das Wort Applet setzt sich aus „application“ und „snippet“ zusammen, was wohl als „kleine Anwendung“ gedeutet werden kann. Applets werden im Vergleich zu Applikationen von der Klasse `java.applet.Applet` abgeleitet und sind dazu ausgelegt, über einen Browser vom Server heruntergeladen und in diesem ausgeführt zu werden. Der Browser ruft einmalig die Methode `init()` auf, in der – ähnlich wie bei der gleich lautenden Methode eines Servlets – initiale Arbeiten getan werden können. Immer wenn das Applet im Browser sichtbar/unsichtbar wird, wird die Methode `start()`/`stop()` aufgerufen. Bei Verlassen der Seite wird `destroy()` aufgerufen, hier können dann z.B. benutzte Ressourcen freigegeben werden. Applets arbeiten komplett clientseitig, während Servlets auf den Server beschränkt sind.

Leider sind Applets auf den sie umgebenden Browser angewiesen. Das macht Applet-Programmierung recht schwierig und mitunter unmöglich. Denn lange nicht jeder Browser unterstützt von Haus aus Java-Applets. So liefert Microsoft seinen Internet Explorer und das Betriebssystem Windows XP gar nicht mehr mit der für Applets notwendigen Java Virtual Machine aus. Es gibt zwar Plugins für viele Browser, doch kann man nicht davon ausgehen, dass jeder Benutzer diese Zusatz-Software installiert hat. Zudem ist es schwierig, solch ein Plugin zu finden, denn es wird so gut wie nie automatisch heruntergeladen und installiert, so wie das bei Macromedias Flash-Plugin vorbildlich funktioniert. Es würde auch ausreichen, ein aktuelles JRE zu installieren, aber welcher Computer-Anfänger findet auf der Sun-Homepage angesichts der vielen Java-Versionen und -Ausprägungen auf Anhieb die für ihn passende Version?! Es kommt noch erschwerend hinzu, dass die unterstützenden Browser unterschiedliche Java-Versionen implementieren (und oft recht alte); Man versucht sich als Entwickler also auf einen minimierten Befehlssatz zu beschränken um möglichst vielen Benutzern die Ausführung des Applets zu ermöglichen.
Viele Browser unterstützen Java immer noch nur in der uralten Version 1.1.x. Deshalb sollten Applets mit dem Flag `-target 1.1` kompiliert werden.

4.5.1 Unprivilegierte Applets

Wenn man von einem Server ein (unbekanntes) Programm herunterlädt, das sich dann automatisch ausführt, muss man dem Server und dessen Inhalten entweder sehr vertrauen, oder man muss sich sicher sein, dass das Programm in seiner Ausführung technisch so beschränkt wird, dass es nichts Böses tun kann. Letzteres ist bei Applets der Fall: Unprivilegierte Applets dürfen beispielsweise auf keine Dateien des Clients zugreifen (auch keine erzeugen) und keine Klassen von anderen Servern laden. Es sind also nur Klassen

erlaubt, die von dem Server stammen, von dem auch das Applet selbst geladen wurde. Applets dürfen keine lokalen Programme ausführen und keine Netzwerkverbindungen zu fremden Rechnern erstellen. Dies alles überwacht ein Sicherheitsmanager.

Der Chat-Client tappte in eine weitere „Falle“: Normale Applets dürfen beim Erstellen des Sockets (um Verbindung mit dem DiLog-Server aufzunehmen) den lokalen Client-Port nicht festlegen (nur zufällig wählen). Sie dürfen Sockets also nur unter Angabe des Servernamens und -ports erstellen (`Socket socket = new Socket(serverName, serverPort);`). Dies ist aber ungeschickt, wenn der Client durch eine Firewall geschützt ist, die er von innen nur auf einigen wenigen festgelegten Ports verlassen darf. Dies ist in gut gesicherten Firmennetzwerken häufig der Fall.

4.5.2 Privilegierte Applets

Damit der Benutzer des Chat-Clients seinen lokalen Port selbst auswählen kann, muss das Socket mit den zusätzlichen Parametern Name und Port des Clients erzeugt werden: `Socket socket = new Socket(serverName, serverPort, clientName, clientPort);`.
Wenn man es so versucht, führt das jedoch zu einer Exception, weil ein unsigniertes Applet den Client-Port nicht frei wählen darf. Interessanterweise ist es keine Security-Exception des SecurityManagers, sondern eine java.net.ConnectException: Connection timed out: connect.

Deshalb wird das Applet signiert (keytool und jarsigner sind Programme des J2SDK):

```
keytool -genkey -alias dilog
keytool -selfcert
jarsigner -verbose /usr/local/tomcat5/webapps/dilog/web/applet/chatapplet.jar dilog
```

Das ist zwar auch wieder nur ein selbsterstelltes Zertifikat, jedoch erfüllt es seinen Zweck (Vereinsmitglieder können ihrem Verein prinzipiell vertrauen). Die Erstellung des Zertifikats und das Signieren des Applets ist als vollständiger Ant-Task (Kap. 4.7) umgesetzt.

4.6 XML

Die Extensible Markup Language ist eine Anfang 1998 vom W3C (World Wide Web Consortium) standardisierte Meta-Beschreibungssprache, über die eigene Tags definiert werden können. Sie wurde von SGML (Standard Generalized Markup Language) abgeleitet und stellt dementsprechend auch eine SGML-Untermenge dar.[45]

XML hat viele Bereiche der Informationsindustrie revolutioniert: So ist man dazu übergegangen, Dokumentationen von Maschinen und Geräten in XML zu gießen, Kataloge werden mittels XML-Datenhaltung produziert und im IT-Bereich sind XML-Konfigurationsdateien selbstverständlich. Dabei definiert XML den grundsätzlichen Aufbau einer Datei. Für konkrete Anwendungen müssen weitere Details definiert werden, z.B. wo welche Tags stehen und welche Attribute wo verwendet werden dürfen.

[45] Der Erfolg auf breiter Front war SGML im Gegensatz zu XML nicht gegönnt. Dafür war SGML einfach zu komplex: Allein die Dokumentation aller Befehle umfasste schon 500 Seiten.

Exkurs: Markup Language

Eine Auszeichnungssprache (Markup Language) dient zur Beschreibung von Daten oder eines Verfahrens um Daten darzustellen. Das Auszeichnen von Texten hat seinen Ursprung in der Druckindustrie. Man unterscheidet zwischen Descriptive Markup Language, welche die Syntax von Daten beschreibt, und Procedural Markup Language, mit der das Verfahren zur Darstellung der Daten beschrieben wird. Mit einer Auszeichnungssprache werden also Eigenschaften, Zugehörigkeiten und Verfahren von Wörtern, Sätzen oder Abschnitten eines Textes beschrieben. Dazu besitzt sie wie alle Sprachen Syntax, Semantik und Grammatik.

Zur ersten Gruppe gehören beispielsweise SGML, XML, HTML, WML und SVG. Prozedurale Auszeichnungssprachen sind dagegen TeX, das bekannte PDF und PostScript. Letztere weisen das Ausgabemedium (z.B. einen Drucker) an, wie der Inhalt ausgegeben werden soll.

Der wesentliche Vorteil von XML/SGML besteht darin, dass Dateninhalt strikt von seiner Repräsentation getrennt ist. Dieser Vorteil soll mit einem kleinen Beispiel zementiert werden. Das Wartungs- und Reparaturhandbuch eines Airbus A320 umfasst ca. 24.000 Seiten (Quelle: Pressearchiv 2002 der Verkehrswerkstatt[46]). Jede größere Firma oder Organisation legt Wert auf ihr CI (Corporate Identity), was sich auch in einem eigenen CD (Corporate Design) manifestiert. Unter anderem deshalb wollen Fluggesellschaften, die das Flugzeug kaufen, diese Handbücher in ihrem eigenen Firmen-Design herausbringen. Der Aufwand für Airbus wäre immens, wenn für jede Fluggesellschaft ein eigenes Handbuch geschrieben werden müsste. Stattdessen werden alle Daten in XML oder SGML hinterlegt. Das Design (Formatierung und Anordnung) dieser Daten kann nun jede Fluggesellschaft durch eine sog. XSL-Transformation[47] selbst vornehmen bzw. anpassen.
Ein weiterer Vorteil an XML-Dateien ist die Lesbarkeit durch Menschen und Maschinen.

4.6.1 XML-Parser

[Seeboerger-Weichselbaum 2002] (S. 326) beschreibt einen Parser wie folgt:

> *Ein Parser ist ein Algorithmus, der zu einem vorgegebenen Satz (Wort) entscheidet, ob dieser zu einer Sprache gehört, die meist durch eine Grammatik implizit vorgegeben ist. Ein Interpreter ist eine Parser-Erweiterung. Neben der Satzanalyse (also dem Ableitungsversuch) erfolgt beispielsweise die Berechnung und Speicherung von Variablenwerten, Befehlen und des ganzen Programms. Der Interpreter berechnet (interaktiv) den »Wert« des Strukturbaums, der von dem Parser geliefert wird.*

Ein XML-Parser hat also die Aufgabe, eine XML-Datei einzulesen und auf die XML-Syntax hin zu überprüfen. Hält die Datei sämtliche Regeln für XML ein, so spricht man von einem wohlgeformten XML-Dokument. Anschließend wird – wenn keine Fehler auftauchten – ein Ableitungsbaum aufgebaut um die eingelesenen Daten weiterverarbeiten zu können.

[46] http://www.bics.be.schule.de/son/verkehr/presse/2002_1/v5421_16.htm

[47] XSL bedeutet Extensible Stylesheet Language. Sie wird dazu benutzt, um mit Hilfe eines XSLT-Prozessors, der XSL-Stylesheets und XML-Dateien einliest, Daten und Layout in einer neuen Ausgabedatei zu vereinen.

Es gibt verschiedene Parser mit unterschiedlichen Ansätzen und Zielen. Bekannt sind der in Java geschriebene Parser XP von James Clark oder der ebenfalls in Java und anderen Sprachen vorliegende Apache-Parser Xerces. Hier zeigt sich ein Nachteil, den JAXP aufzuheben versucht: Jeder Parser benutzt eine eigene API, d.h. die Parser können nicht einfach ausgetauscht werden, da sie oft nicht vollständig kompatibel zueinander sind.

4.6.2 XML und DTD

Oft ist es zweckmäßig, wenn das XML-Format bzw. die Struktur mittels einer DTD (Document Type Definition)[48] definiert wird. Damit kann man erreichen, dass eine XML-Datei nicht beliebige Tags definiert und verwendet. Dies ist z.B. immens wichtig, wenn zwei Applikationen untereinander Daten austauschen. Hält sich das XML-Dokument an die in der zugehörigen DTD aufgestellten Regeln, so spricht man von einem gültigen XML-Dokument. Um dies zu demonstrieren nachfolgend die DTD zur XML-Datei `boatconf.dilog.xml`, welche die dem System zur Verfügung stehenden Boote definiert.

```
<!-- mindestens 1 Tag boat -->
<!ELEMENT boats (boat+)>

<!-- 1x Tag mysqlname, 1x Tag accessname, beliebig oft oder gar nicht Tag equalboat -->
<!ELEMENT boat (mysqlname, accessname, equalboat*)>

<!-- Attribut id (Text) zwingend benoetigt -->
<!ATTLIST boat id CDATA #REQUIRED>

<!-- Textinhalt -->
<!ELEMENT mysqlname (#PCDATA)>

<!-- Textinhalt -->
<!ELEMENT accessname (#PCDATA)>

<!-- Textinhalt -->
<!ELEMENT equalboat (#PCDATA)>
```

Listing 11: DTD boatconf.dilog.dtd

Die DTD legt die Reihenfolge und Art der in der XML-Datei verwendbaren Elemente und der jeweils zugehörigen Attribute fest. Mit `ELEMENT` werden Elemente definiert (die weitere Elemente enthalten können), welche `boatconf.dilog.xml` verwenden kann oder muss. `ATTLIST` definiert Attribute zu einem Element. Spezielle Zeichen hinter den Elementnamen geben an, wie oft ein Element verwendet werden darf oder muss (siehe Kommentare in Listing 11). Die DTD sagt im Wesentlichen aus, dass die XML-Datei mindestens ein Tag `boat` (mit einer `id` als Attribut) enthalten muss, welches die geschachtelten Tags `mysqlname` und `accessname` enthalten muss und beliebig viele Tags `equalboat` enthalten darf. Der Modifizierer `#REQUIRED` schreibt vor, dass dieses Attribut zwingend angegeben werden muss. `CDATA` (Akronym für Character Data) und `PCDATA` (Parsed Character Data) weisen auf den Datentyp Text hin.

Das folgende Listing 12 zeigt, wie die XML-Datei `boatconf.dilog.xml` die DTD `boatconf.dilog.dtd` einbindet (2. Zeile) und verwendet.

[48] [Matzke 2003] (S. 15) spricht von Data Type Definition. Obwohl das W3C von Document Type Definition spricht, wird diese abweichende Form laut Internet-Suchmaschine Google immerhin ca. 18.000 Mal im Internet benutzt (gegenüber ungefähr 274.000 Treffer für Document Type Definition), Stand 06.01.2005.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE boats SYSTEM "dtd/boatconf.dilog.dtd">

<boats>

   <boat id = "3">
      <mysqlname>Toke</mysqlname>
      <accessname>Toke</accessname>
      <equalboat>4</equalboat>
      <equalboat>17</equalboat>
   </boat>

   <boat id = "4">
      <mysqlname>Toke_Auflage</mysqlname>
      <accessname>Toke saisonbeschränkt</accessname>
      <equalboat>3</equalboat>
      <equalboat>17</equalboat>
   </boat>

   <boat id = "17">
      <mysqlname>Toke_Test</mysqlname>
      <accessname>Toke Spinnacker</accessname>
      <equalboat>3</equalboat>
      <equalboat>4</equalboat>
   </boat>

</boats>
```

Listing 12: XML-Konfigurationsdatei boatconf.dilog.xml

4.6.3 SAX und DOM

Seit Java 1.4 ist XML in das J2SDK integriert, d.h. es müssen für Projekte, die XML verwenden, keine zusätzlichen JAR-Bibliotheken mehr eingebunden werden. Java-Applikationen können nun ohne Zusatzaufwand XML-Dateien einlesen, auswerten und erstellen. Dies wird durch die sog. Java API for XML-Processing (JAXP) ermöglicht, die in Java 1.4 als Version 1.1 vorliegt. Java und XML gehen dabei eine Idealehe ein: Java ermöglicht plattformunabhängige Programme, während XML eine plattformunabhängige Datenstruktur bereitstellt. JAXP enthält zwei Standard-APIs, nämlich SAX (Simple API for XML) und DOM (Document Object Model). Konkret implementiert wird JAXP mit dem Java-XML-Parser Crimson vom Apache XML Project.

DOM bildet die immer hierarchisch aufgebaute XML-Datei intern als Dokumentenbaum ab. Dieser Baum besteht aus unterschiedlichen Objekten, die manipuliert werden können, bevor sie weiterverwendet oder in eine XML-Datei zurückgeschrieben werden. Die einzelnen Elemente des Baumes stehen zueinander in Beziehung (als Eltern, Geschwister oder Kinder). Zum Bilden des Dokumentenbaumes muss logischerweise die gesamte XML-Datei eingelesen werden, was bei großen XML-Dateien ein Problem darstellt, denn der Speicherbedarf wächst proportional zur Dateigröße und der Vorgang ist zudem rechenintensiv, und damit also auch zeitintensiv.

Dies führte zur Entwicklung von SAX. Im Gegensatz zu DOM ist SAX nicht vom W3C standardisiert, wird aber de facto als Standard betrachtet. Die Arbeitsweise unterscheidet sich grundlegend von DOM: Zwar wird natürlich auch die Wohlgeformtheit des XML-Dokuments überprüft, aber kein Dokumentenbaum erzeugt. Die XML-Datei wird zeilenweise Tag um Tag eingelesen und danach wieder vergessen. Dabei werden verschiedene Ereignisse ausgelöst, welche die Abarbeitung von vom Entwickler definierten Aufgaben

anstößt. Dadurch, dass SAX sequentiell vorgeht und keine XML-Tag-Informationen oder Inhalte speichert, ist es sehr schnell und benötigt wenig Speicher.

DiLog verwendet DOM. Rein theoretisch wäre bei derzeitigem Projektstand auch die Verwendung von SAX durchaus ausreichend, denn es wird nur lesend auf verschiedene Konfigurationsdateien zugegriffen. Es ist jedoch denkbar, dass bei einer Weiterentwicklung von DiLog die Wartung dieser Konfigurationsdateien über eine grafische Weboberfläche erledigt werden könnte. Dann müsste auch eine Methode zur Änderung und Speicherung von Konfigurationswerten eingebaut werden; Spätestens dann würde sich DOM als notwendig erweisen. Das Verwenden von DOM schon jetzt hat keine Nachteile, da die Konfigurationsdateien so klein sind, dass sich keine Zeit- oder Geschwindigkeitsnachteile bemerkbar machen können.
Zum Datenaustausch mit dem Import-Export-Tool wird ebenfalls DOM verwendet, da hier Daten geschrieben werden müssen. Diese Aufgabe ist schon relativ zeitintensiv, wobei nicht geprüft wurde, wie sich der Zeitverbrauch auf Datenbankzugriffe und XML-Behandlung der gelesenen oder zu schreibenden Daten verteilt.

4.6.4 Parserproblem in DiLog

Beim Erstellen der XML-Datei zum Datentransfer zwischen den Datenbanken MySQL und Access war das seltsame Phänomen zu beobachten, dass derselbe Code mit derselben Java-Version ausgeführt zu unterschiedlichen Ergebnissen führte. Der Java-Code (hier auszugsweise) war korrekt:

```
Element root = document.createElement("database");
root.setAttribute("timestamp", timestamp + "");
document.appendChild(root);
```

In Eclipse, ausgeführt als Applikation, wurde die Datei korrekt erzeugt (hier der Dateibeginn):

```
<?xml version='1.0' encoding='ISO-8859-1' ?>

<database timestamp="1090357150647">
<table name="Mitglieder">
<row>
<column name="Nummer">1
</column>
<column name="New">false
</column>
```

Als Servlet im Servlet-Container des Tomcat-Servers wurde ohne weitere Fehlermeldung nur der Beginn der Datei erstellt:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
[database: null]
```

Eher durch Zufall modifizierte ich den korrekten Source Code so, dass er nun zwar fehlerhaft, aber syntaktisch immer noch korrekt war:

```
Element root = document.createElement("database");
Attr a = document.createAttribute("testattr");
root.appendChild(a); // Fehler
document.appendChild(root);
```

Die Methode `appendChild(Node newChild)`, definiert im Interface `Node`, akzeptiert (rein syntaktisch) als neues Kind alle Objekte, die vom Typ `Node` sind. Da Objekte des Typs `Element` durch die Ableitung von `Node` auch Nodes sind, kann man einem `Element` mit `appendChild()` ohne Probleme ein weiteres `Element` hinzufügen. Attribute, repräsentiert durch das Interface `Attr`, erben auch von `Node` und sind somit also auch Nodes.
Nun werden einem `Element` Attribute aber mit der Methode `setAttribute()` hinzugefügt und nicht mit `appendChild()`, obwohl dies syntaktisch möglich wäre. Der Compiler kann

diesen Fehler nicht erkennen, da das Interface `Attr` wie Interface `Element` vom Interface `Node` abgeleitet ist. Folgerichtig wird ein Laufzeitfehler entdeckt (hier in Eclipse ausgeführt):

```
org.apache.crimson.tree.DomEx: HIERARCHY_REQUEST_ERR: Dieser Knoten ist dort nicht
erlaubt. at org.apache.crimson.tree.ElementNode2.checkChildType(ElementNode2.java:258)
at org.apache.crimson.tree.ParentNode.appendChild(ParentNode.java:341)
```

Interessanterweise sah die Fehlermeldung bei Verwendung von Tomcat anders aus:

```
org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR: An attempt was made to insert a node
where it is not permitted. at org.apache.xerces.dom.ParentNode.internalInsertBefore
(Unknown Source) at org.apache.xerces.dom.ParentNode.insertBefore(Unknown Source)
at org.apache.xerces.dom.NodeImpl.appendChild(Unknown Source)
```

Der korrekte Quell-Code funktionierte also deshalb nicht korrekt, weil eine Parser-Inkompatibilität vorlag. Während in Eclipse der in Java integrierte Crimson-Parser zum Einsatz kam, bevorzugt Tomcat in Version 5 standardmäßig den Parser Xerces. Um Xerces auszuschalten mussten im Verzeichnis `tomcat5\common\endorsed` die Dateien `xercesImpl.jar` und `xmlParserAPIs.jar` gelöscht werden. Danach wurden beim fehlerhaften Code dieselben Exceptions angezeigt, sprich dieselben Parser verwendet. Nun konnte auch mit Tomcat wie in Eclipse eine vollständige XML-Datei erstellt werden.

4.7 Build-Tool Ant

Der Klappentext von [Edlich 2003] beschreibt Ant kurz und treffend:

> *Ant ist ein neues Build-Werkzeug für die Java-Anwendungsentwicklung, das plattformübergreifend konzipiert ist und als Format für Buildfiles XML verwendet. Für Java-Programmierer ist Ant eine gute Alternative zu make, da das Tool speziell auf die Erfordernisse des Build-Prozesses in der Java-Umgebung zugeschnitten ist. Ant wurde im Rahmen des Jakarta-Projekts der Apache Software Foundation entwickelt und ist Open Source.*

Das Buildfile `buildDiLog.xml` wird auf der Kommandozeile aufgerufen. Dabei wird als Parameter ein Target-Name übergeben. Dieses Target wird dann ausgeführt. `ant -f buildDiLog.xml javadoc` erstellt zum Beispiel die JavaDoc-Dokumentation. Ein Target enthält verschiedene Anweisungen und kann auch auf andere Targets verweisen, die vor Abarbeitung des aufgerufenen Targets ausgeführt werden (z.B. ein Backup erstellen vor Neukompilierung).

DiLog benutzt Ant im Prinzip für alle während der Software-Entwicklung anfallenden Aufgaben: JavaDoc erstellen, Source Code kompilieren, Verzeichnisse löschen und erstellen, Dateien kopieren, JSP-Syntaxprüfung, WAR-Dateien fürs Deployment generieren, Datenbanktabellen kopieren oder MySQL-Dumps erstellen, Applet-Dateien für Java 1.1 kompilieren und in ein JAR verpacken, Schlüsselpaar erstellen und Applet signieren, die JAR-Dateien für die Applikationen IETool und Webcam-Client generieren und über den Tomcat-Manager die Webapplikation DiLog neuladen. Viele Aufgaben werden in Abhängigkeit des Zielsystems (Entwicklungsumgebung, Testumgebung oder Produktionsumgebung) erledigt, was über die Konfigurationsdatei `dilog.properties` gesteuert wird.

Die Datei `buildDiLog.xml` ist im Anhang abgedruckt.

4.8 Logging mit Log4J

Log4J ist ebenfalls Open Source Software und kam 1996 auf. Offensichtlich war Log4J erfolgreich, denn es wurde nach C, C++, C#, Perl, Python, Ruby und Eiffel portiert. Log4J wird – wie der Name schon sagt – für Logging in Applikationen verwendet. Abhängig von der einstellbaren Layout-Klasse schreibt jeder Log-Befehl beim Aufruf gewisse Informationen in eine Logdatei. Mit ihrer Hilfe kann ein Entwickler im Falle eines aufgetauchten Fehlers die Abläufe nachvollziehen und leichter feststellen, wie es dazu kam. Je nach `Appender` können die Logausgaben auf die Konsole, in eine Datei oder in GUI-Komponenten geschrieben werden. Zudem sind mehrere Appender für getrennte Logdateien möglich (eine für DiLog und eine für das Chat-Applet).

Der `Logger` wird mit dem Namen der jeweiligen Klasse initialisiert (siehe 1. Anweisung der Klasse `UploadServlet` in Listing 4). Für die Konfiguration von Log4J wird entweder die Klasse `DOMConfigurator` (benutzt eine XML-Datei als Konfigurationsdatei) oder die Klasse `PropertyConfigurator` (benutzt eine .properties-Datei zur Konfiguration) verwendet. Der Pfad zu der mit der `configure()`-Methode übergebenen Datei kann absolut oder relativ angegeben werden. Es existieren verschiedene Loglevel: debug, info, warn, error, fatal. In der Konfigurationsdatei kann der Level gesetzt werden. Bei Level fatal werden beispielsweise nur `logger.fatal()`-Aufrufe geloggt, bei Level debug dagegen alle.

Die Initialisierung in Webanwendungen erfolgt zwangsläufig anders. Hier wird in die Datei `web.xml` (siehe Anhang) eine Servlet-Klasse eingetragen, welche die oben aufgeführten Initialisierungen durchführt. Dieses Servlet wird dann automatisch mit dem Start des Servlet-Containers ausgeführt.

Logging verursacht zwar Kosten bei der Ausführung eines Programms, jedoch fällt das während der Entwicklung nicht ins Gewicht und für ein stabiles Produktivsystem kann der Loglevel erhöht werden, sodass Debug-Informationen gar nicht erfasst werden.
Weitere Informationen und Beispiele befinden sich im PDF „Vorgehen Log4J“ auf der CD.

4.9 Testen mit JUnit

JUnit[49] ist ein hauptsächlich von Erich Gamma[50] und Kent Beck entwickeltes Test-Framework, mit dem man hauptsächlich Klassen (sog. Units) automatisiert testen kann. Es basiert auf Konzepten, die schon für Smalltalk entwickelt wurden. JUnit lässt sich mit eigener GUI in Eclipse einbinden.

Ein JUnit-Test hat als Ergebnis entweder das Gelingen oder das Misslingen des Tests. Dazwischen gibt es nichts. Dieses Schwarz-Weiß-Denken (bzw. rot und grün) ist deshalb angebracht, weil das Testergebnis vom Entwickler bzw. Tester vorgegeben wird. Wenn das vorgegebene Testergebnis dem tatsächlichen Ergebnis entspricht, war der Test erfolgreich.

[49] http://www.junit.org

[50] Gamma hat mit den anderen drei Mitgliedern der legendären Gang of Four das bekannte Buch „Design Patterns“ geschrieben.

```
package magtime.testing.junit;

import junit.framework.TestCase;
import magtime.struts.helper.BaseActionForm;

/**
 * Diese JUnit-Testklasse testet die Prüfmethoden der Klasse BaseActionForm.
 * BaseActionForm stellt Methoden bereit um Formularfelder auf bestimmte
 * Eigenschaften und eingegebene Zeichen zu überprüfen.
 * @author Steffen Hartmann
 */
public class BaseActionFormTest extends TestCase {

   String nullString;
   String emptyString;
   String testString;
   BaseActionForm baf = new BaseActionForm();

   public void testIsEMail() throws Throwable {

      // erwarteter Wert, tatsächlicher Wert
      assertEquals(false, baf.isEMail(emptyString));
      assertEquals(false, baf.isEMail(testString));
      assertEquals(false, baf.isEMail("123"));
      assertEquals(false, baf.isEMail(" "));
      assertEquals(false, baf.isEMail("sh*@hdm-stuttgart.de"));
      assertEquals(false, baf.isEMail("s20hdm.de"));
      assertEquals(false, baf.isEMail("test@em#.de"));
      assertEquals(false, baf.isEMail("auchtest@sdfsdfd"));
      assertEquals(false, baf.isEMail("@sdfsdf.com"));
      assertEquals(true, baf.isEMail("steffen.hartmann@msphartmann.de"));
      assertEquals(true, baf.isEMail("webmaster@segelecke.de"));
      assertEquals(true, baf.isEMail("m.h.loy@test.org"));
      assertEquals(true, baf.isEMail("mastercard_test@hh.kredit.info"));
      assertEquals(true, baf.isEMail("mastercard_test@hh.kredit.stg.info"));
      assertEquals(true, baf.isEMail("a32@hm-21.com"));
      assertEquals(true, baf.isEMail("do_21@web-de.de"));
      assertEquals(true, baf.isEMail("isOk@ok.ds"));
      assertEquals(false, baf.isEMail("+"));

   }

   public void testIsEmpty() throws Throwable {

      assertEquals(true, baf.isEmpty(emptyString));
      assertEquals(true, baf.isEmpty(nullString));
      assertEquals(false, baf.isEmpty(testString));
      assertEquals(false, baf.isEmpty("123"));
      assertEquals(false, baf.isEmpty(" "));
      assertEquals(false, baf.isEmpty("+"));

   }

}
```

Listing 13: Auszug aus der JUnit-Testklasse BaseActionFormTest

Eine Testklasse erweitert die JUnit-Klasse `junit.framework.TestCase`. Die Testklasse bekommt den Namen der zu testenden Klasse mit angehängtem „Test“. Nun werden die Testmethoden geschrieben. Sie sollen gleich heißen wie die zu testenden Methoden, zusätzlich wird das Präfix „test“ vorangestellt.

Es stehen verschiedene Assert-Methoden zur Auswahl (`assertEquals()`, `assertNotNull()`, `assertNull()`, `assertSame()` und neuerdings auch `assertTrue()`, welches erst in einer der neueren Versionen hinzukam). Als erster Parameter wird der erwartete Wert übergeben, der zweite ruft die zu testende Methode auf, welche dann natürlich einen Wert zurückgeben muss. In Listing 13 wurde die Klasse `magtime.struts.helper.BaseActionForm` mit allen ihren Methoden getestet. Sie ist die Basisklasse für alle Struts-ActionForm-Klassen und stellt Methoden zur Überprüfung der vom Benutzer in ein Formular eingetragenen Werte zur Verfügung.

Die im Listing nicht gezeigten Methoden

```
protected void setUp() throws Exception {
   testString = "Dies ist ein Test-String";
   nullString = null;
   emptyString = "";
}

protected void tearDown() throws Exception {
   super.tearDown();
}
```

initialisieren häufig benutzte Variablen mit gleichbleibenden Werten. `tearDown()` dient dazu, evtl. in `setUp()` verwendete Ressourcen wieder freizugeben.

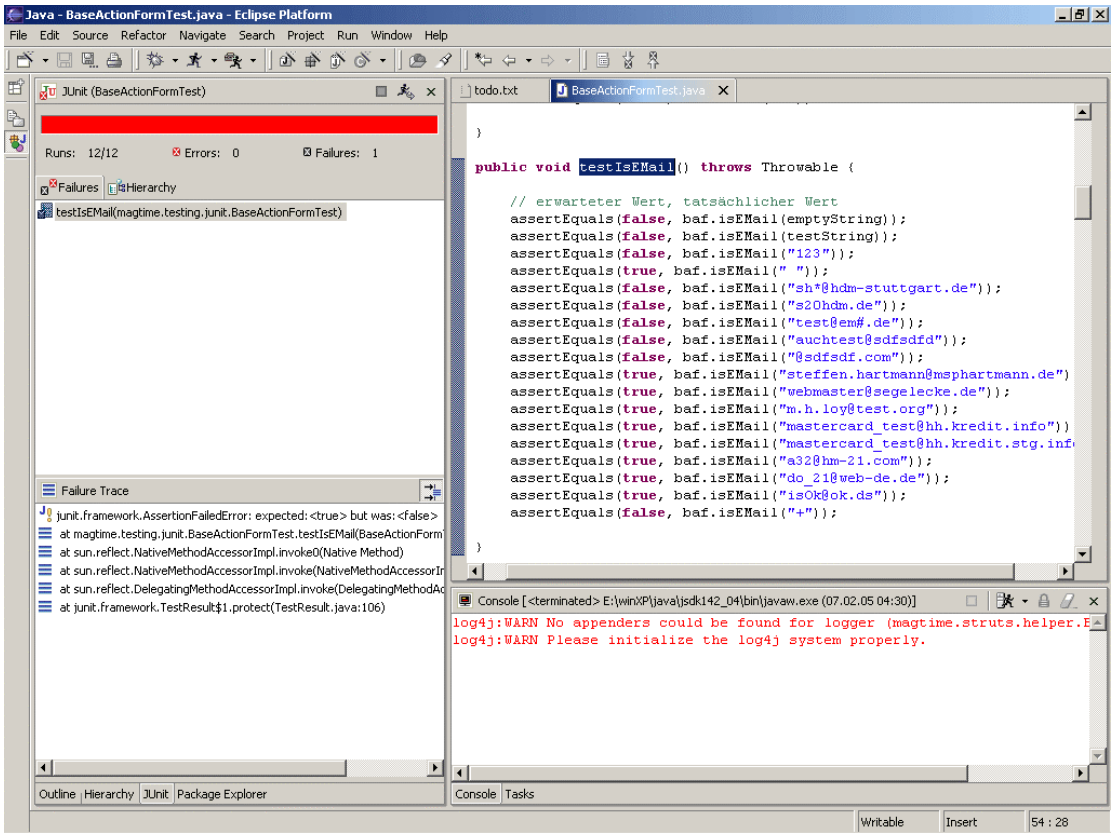


Abb. 12: Java-Entwicklungsumgebung Eclipse mit fehlgeschlagenem JUnit-Test

5 Funktionsbeschreibung des Digitalen Logbuchs

In diesem Kapitel werden einige (formatfüllende) Use-Case- und Activity-Diagramme der UML verwendet um die Funktionalität der Software zu verdeutlichen. Oftmals sprechen sie dermaßen für sich, dass kaum noch weitere erklärende Sätze nötig sind.

5.1 Übersicht

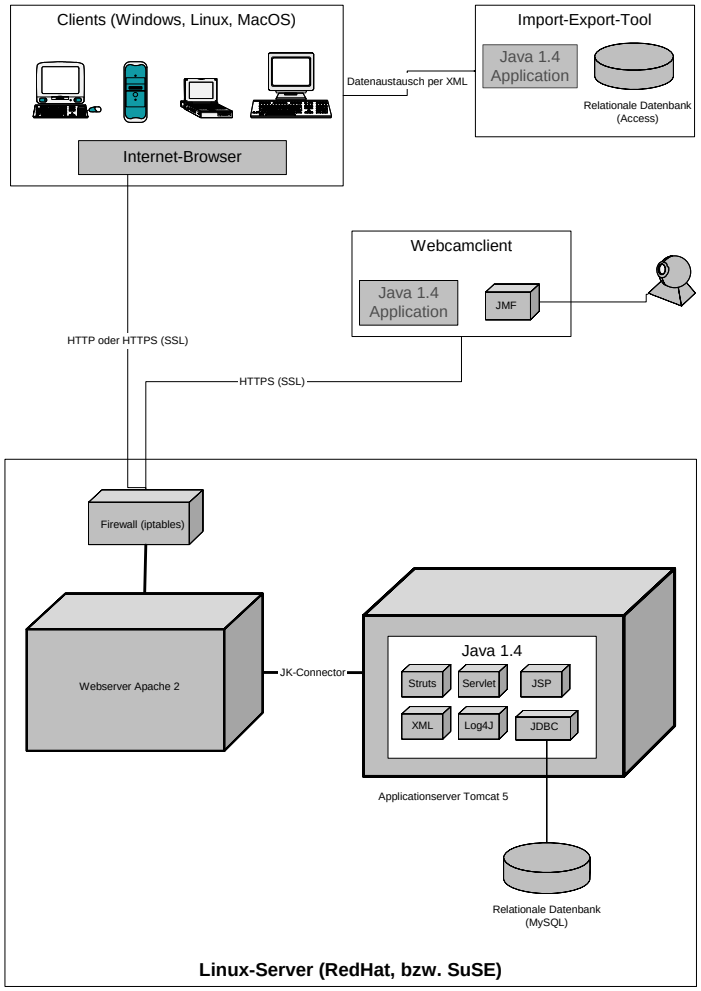


Abb. 13: Gesamtarchitektur DiLog (Komponentensicht)

DiLog kommt in drei separaten Software-Paketen daher:

- Hauptbestandteil von Dilog sind die serverseitigen Artefakte (dieses und das darauf folgende Kapitel).
- Die lokal beim Administrator installierte Java-Applikation Import-Export-Tool (IETool) wird in Kapitel 7 erläutert.
- Der Webcam-Client (Kapitel 8) kann ebenfalls als Java-Applikation in der Nähe des Segelreviers auf einem PC mit Internet-Anschluss installiert werden.

Das Hauptpaket besteht im Wesentlichen aus Java-Dateien, Java Server Pages, XML-Dateien und Bildern. Zusätzlich gibt es mehrere DTDs, eine CSS- und eine HTML-Datei plus zusätzliche Properties-Dateien. Eingebettet sind diese Dateien alle im Servlet-Container Tomcat. Dort befinden sie sich in Gesellschaft mit einigen unterstützenden Java-Bibliotheken, APIs und Frameworks, wovon ein paar in Abb. 13 zu sehen sind. Des Weiteren existiert die Datenbank MySQL und der Webserver Apache.

Benutzer können sich über einen Browser ihrer Wahl auf dem DiLog-Server[51] einloggen und verschiedene Geschäftsprozesse in Gang setzen. Der allgemeine Benutzer kann entsprechend seinen Benutzerrechten in weitere Kategorien aufgeteilt werden: Gast,

[51] DiLog-Server meint sowohl den Webserver Apache und den Applicationserver Tomcat als auch den physikalischen Server, auf dem die beiden Server installiert sind.

Benutzer (auch als User bezeichnet), Obmann und Administrator. Dabei sind die Rechte streng hierarchisch im Sinne einer Spezialisierung vererbt, wobei der Gast als „kosmische Superklasse“ betrachtet werden kann, der Administrator nach Benutzer und Obmann in der Vererbungshierarchie ganz unten steht und demzufolge die meisten (alle) Rechte hat. Auf die unterschiedliche Darstellung der Weboberfläche in Zusammenhang mit den Benutzerrechten wurde in Kapitel 4.4.5 bereits eingegangen. Der Sinn und Zweck dieser Software wurde mit der Aufgabenstellung in Kapitel 1.2 verdeutlicht.

5.2 Datenhaltung und -synchronisierung

Ohne Datenpersistenz ist keine vernünftige Applikation denkbar. Das System muss logischerweise Benutzerdaten, Logindaten, Bootsdaten und Buchungsdaten über eine Session hinaus bewahren. Dabei hängt viel von der Datenstruktur ab; ein ungeschicktes Datenbank-Design kann Flexibilität verhindern, Performance verringern oder Programm-Code aufblähen.

Beim ASV Hamburg bestand nun die Schwierigkeit, dass eine völlig unnormalisierte Access-Datenbank im Sinne eines Legacy-Systems parallel zur MySQL-Datenbank mit in das zukünftige System eingebunden werden musste. Sobald zwei Datenbanken im Spiel sind, die dieselben Daten verwalten sollen, stellt sich natürlich sofort die Frage nach der Synchronisation der Daten. Es musste ein Weg gefunden werden, wie beide Datenbanken ohne großen Aufwand und zuverlässig auf denselben Datenstand gebracht werden können.

Der Einfachheit halber mussten eine Master-Datenbank und eine Slave-Datenbank definiert werden. Per Definition soll die Master-DB die aktuell(er)en Daten enthalten. Da DiLog auch ohne Access funktionieren muss, war die logische Konsequenz, die MySQL-DB über die Access-DB zu erheben. Wenn nun also mit der Access-DB gearbeitet werden muss, so müssen die aktuellen Daten vorher aus der Online-Datenbank heruntergeladen werden. Dafür ist eine Export-Funktion vorgesehen, welche alle Benutzerdaten, die Access benötigt, in eine XML-Datei schreibt und komprimiert. Durch die Datenkompression kann die XML-Transportdatei auf ein Zehntel ihrer ursprünglichen Größe gebracht werden und rangiert somit im zweistelligen Kilobyte-Bereich bei ca. 500 Vereinsmitgliedern. Dies ist selbst für analoge Modems ein unproblematischer Wert.

Die XML-Datei wird vom Import-Export-Tool eingelesen, die Mitgliederdaten werden verarbeitet und in die Access-DB geschrieben. Dabei wird die vorhandene Tabelle gelöscht, eine neue generiert und die Daten eingetragen. Wenn nur Abfragen auf die Access-DB gemacht werden müssen, ist der Austauschprozess hier beendet. Es können jedoch auch Benutzer neu angelegt, gelöscht oder deren Daten verändert werden. Dabei ist entscheidend, dass die jeweiligen Datensätze mit einem speziellen Flag markiert werden (neu, geändert, zu löschen).
Mit demselben IETool können nun alle geänderten Datensätze wiederum in eine XML-Datei exportiert werden. Da hier meist nicht besonders viele Datensätze exportiert werden müssen, beläuft sich die Größe der komprimierten Datei meist auf 1-10 KB, ist also nicht der Rede wert. Die Datei kann nun auf den DiLog-Server hochgeladen und in MySQL importiert werden.

Zu beachten ist bei diesem Import-Export-Zyklus, dass nach Export der Mitgliederdaten aus MySQL die entsprechenden Tabellen gesperrt sind (Schreibschutz). Diese Sperre wird automatisch nach dem Datenimport der in Access veränderten Daten aufgehoben, sie kann aber auch manuell aufgehoben werden, falls ein Datenupload gar nicht stattfinden soll (weil in Access keine Daten verändert wurden). Mit der Sperrung kann erreicht werden, dass nicht schon Änderungen in MySQL gemacht werden, obwohl in Access geänderte Daten noch gar nicht hochgeladen und wieder in MySQL importiert wurden. Datenkonsistenz ist somit also stets gewährleistet.

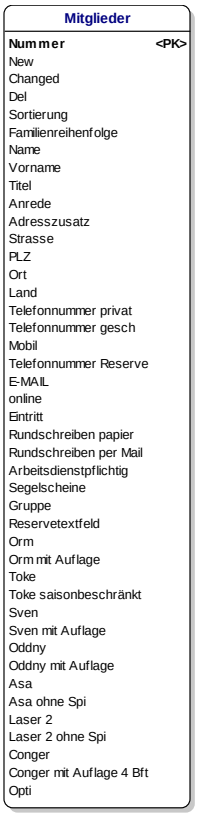


Abb. 14: DB-Schema Access

5.2.1 Datenbank-Design Access

Abbildung 14 zeigt die „historisch gewachsene“ Access-Datenbank, die aus einer einzigen Tabelle mit dem Namen Mitglieder besteht. Sie kann grob in drei Bereiche untergliedert werden: persönliche Daten des Mitglieds, Vereinsdaten des Mitglieds und seine Segelberechtigungen. Es ist klar, dass diese Relation, die ja nun in MySQL abgebildet werden sollte, normalisiert werden muss. Dabei wurde aber nicht besonders streng vorgegangen.

5.2.2 Normalisierung

Um Redundanzen (doppelte Einträge) und Anomalien (sich widersprechende Dateninhalte) zu vermeiden muss das Datenmodell einer relationalen Datenbank normalisiert werden. Dadurch vereinfacht sich auch die Wartung und Datenkonsistenz kann besser gewährleistet werden. Wenn das Modell einen bestimmten Zustand erreicht hat, indem es festgelegte Regeln einhält, spricht man von einer Normalform. Die 1. Normalform verlangt, dass jedes Attribut atomar, sprich unteilbar ist. Dies ist im MySQL-Schema bis auf eine Ausnahme der Fall (Straße fasst Straßenname und Hausnummer zusammen). Die 2. Normalform wurde insofern erreicht, als dass Redundanzen weitestgehend vermieden und Informationen in für diese Applikation sinnvolle Einheiten aufgeteilt wurden.

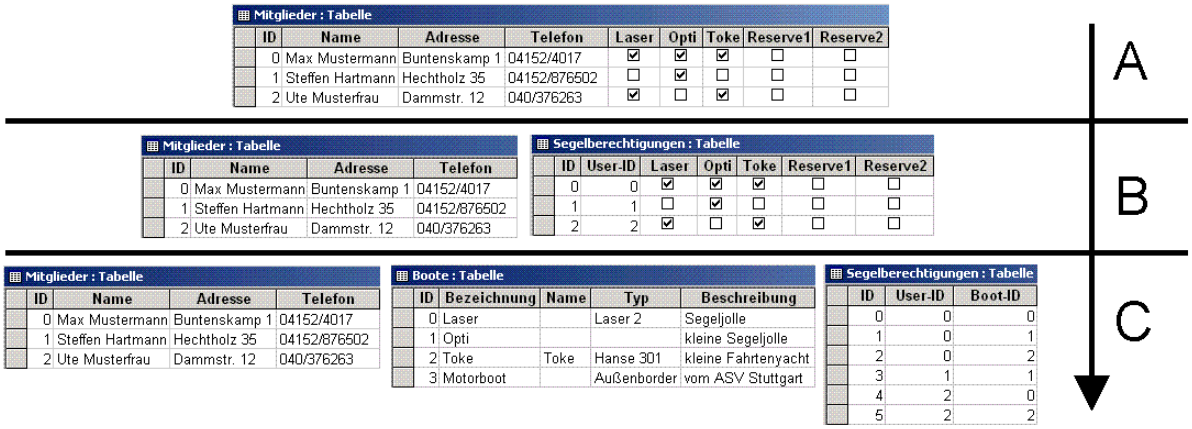


Abb. 15: Normalisierungsbeispiel Segelberechtigungen

In Abbildung 15 zeigt Ebene A die gänzlich unnormalisierte Datenbank, wie sie in Access verwendet wird. Ebene B enthält die nun geteilten und über die ID zusammenführbaren Tabellen. Obwohl die Segelberechtigungen nun getrennt sind von den Personendaten, bleibt ein Nachteil: Die Struktur der Tabelle ändert sich durch eine neue Spalte, wenn ein neues Boot eingefügt wird. Um dies zu vermeiden müssen die Bootsbezeichnungen weg aus dem Spaltennamen. Deshalb führt Lösungsansatz C eine eigene Tabelle Boot ein, und Tabelle Segelberechtigungen stellt die Bezüge zwischen Boot und Mitglied her. Somit können nun Segelberechtigungen hinzugefügt oder gelöscht und neue Boote ohne Code-Anpassungen eingefügt werden. Beim Löschen eines Bootes muss allerdings beachtet werden, dass vorhandene Segelberechtigungen nicht plötzlich ins Leere zeigen. Diese Dateninkonsistenz verhindert der sog. Foreign Key Constraint, der das Löschen in einem solchen Fall verbietet.

5.2.3 Datenbank-Design MySQL

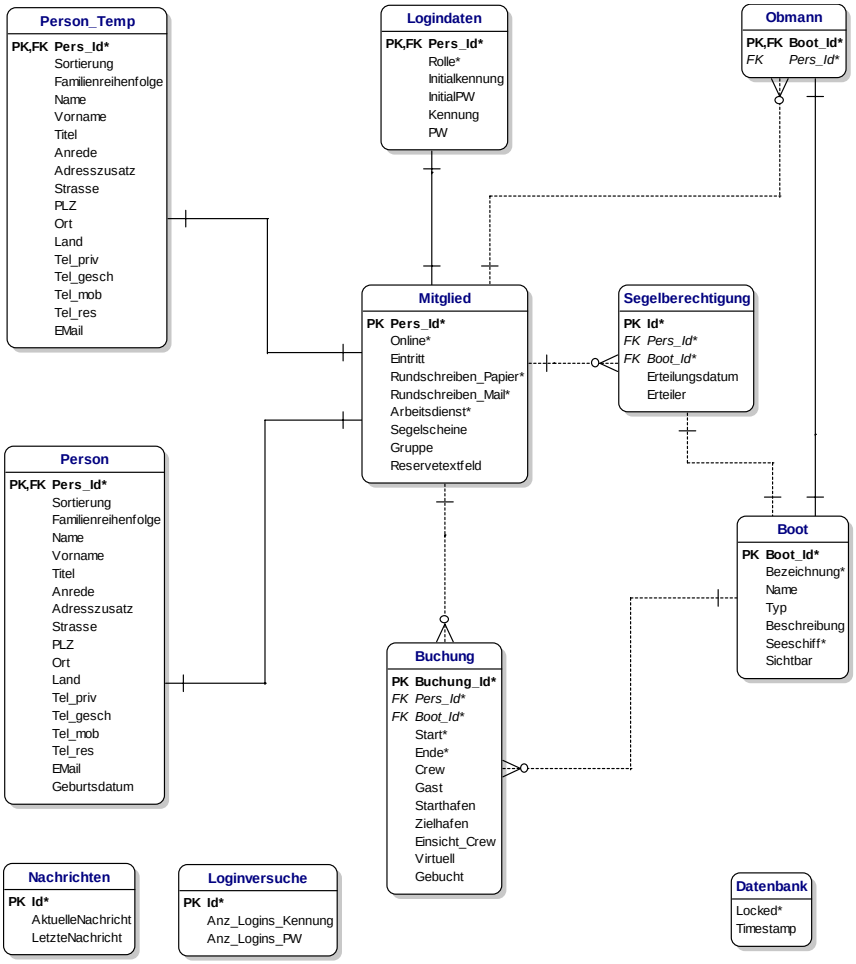


Abb. 16: DB-Schema MySQL

Das Datenbankschema ist selbsterklärend. Die Tabellen Mitglied, Person und Segelberechtigung werden mit dem IETool synchron mit Access gehalten.

5.3 Funktionsbeschreibung Gastzugang

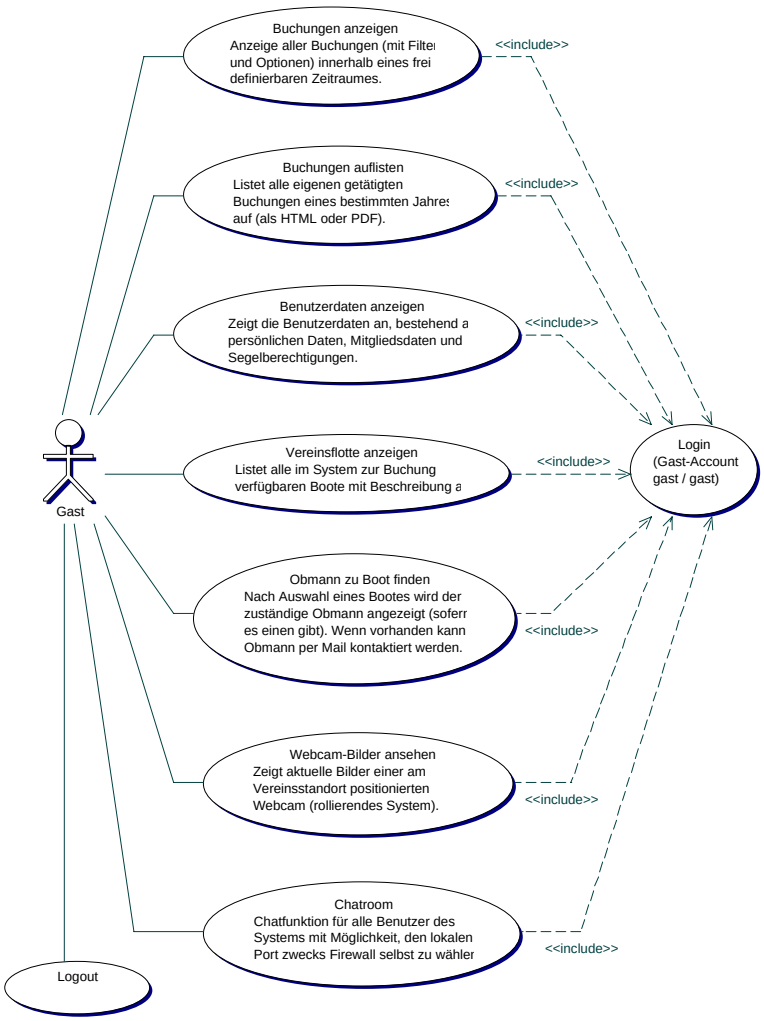


Abb. 17: Funktionsbeschreibung Gastzugang

Das Use-Case-Diagramm (Abbildung 17) beschreibt die dem Benutzer mit Status Gast erlaubten Aktionen. Die Aktion „Buchungen auflisten“ zeigt eine Buchung zu Demonstrationszwecken an, die beim normalen Betrieb von Usern (Mitglieder als Benutzer) nicht gesehen werden kann. Die Benutzerdaten sind ebenfalls nur simuliert und ergeben keinen Sinn. Sie werden bei einem Datenexport von MySQL zu Access entfernt.

Die Webcam-Seiten zeigen zur Zeit pro Seite jeweils neun aktuelle Bilder, wobei das jeweils aktuellste Bild rechts unten platziert ist. Wenn ein neueres Bild vom Webcam-Client auf den DiLog-Server geladen wird, wandern alle Bilder um eine Position weiter. Das letzte älteste Bild auf der zweiten Seite fällt dann raus und wird gelöscht. Der Chatroom kann zur besseren Akzeptanz des Gesamtsystems beitragen. Hier kann ein Austausch zwischen Vereinsmitgliedern und Gästen oder ein das Vereinsleben und den Zusammenhalt fördernder Plausch stattfinden. Zum Chatten muss der Browser Java-Applets unterstützen (siehe Kapitel 4.5).

5.4 Funktionsbeschreibung Benutzerzugang

Wie im Use-Case-Diagramm unschwer zu erkennen ist, erbt der Benutzer alle Funktionen vom Gast und erweitert diese mit vier expliziten Benutzermethoden. Alle Methoden setzen voraus, dass sich der User mit seinen Logindaten (Benutzerkennung und Benutzerpasswort) am System authentifiziert. Diese Logindaten kann der User jederzeit ändern, indem er seine alte und die neue Kennung und/oder sein altes und das neue Passwort angibt. Seine persönlichen Daten kann der Benutzer ebenfalls ändern. Und er kann natürlich – als wesentlicher Bestandteil der Software – Buchungen tätigen und später auch bearbeiten.

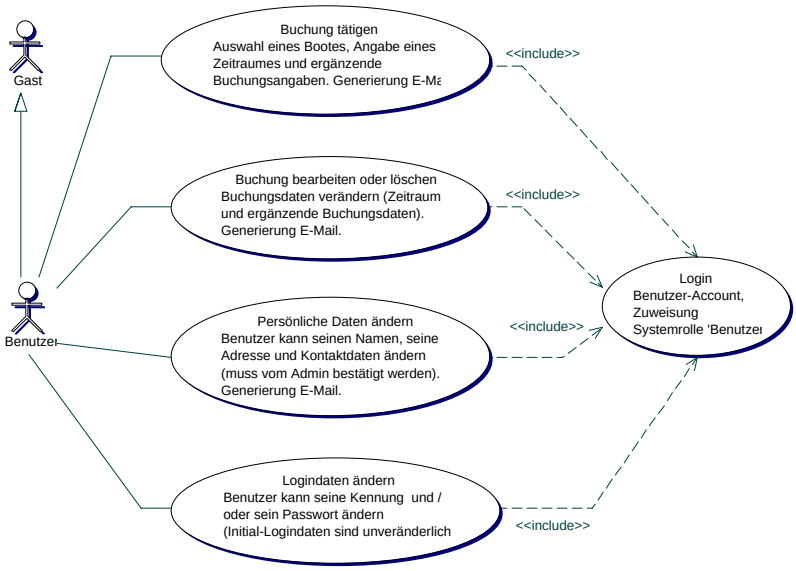


Abb. 18: Funktionsbeschreibung Benutzerzugang

5.4.1 Datenänderungen

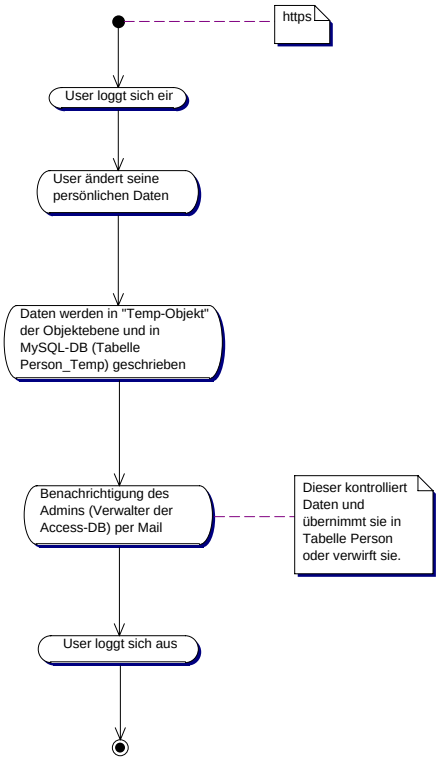


Abb. 19: User ändert persönliche Daten

Bei einigen hundert Mitgliedern ändern sich bald jeden Tag persönliche Daten, sei es ein Adresswechsel durch Umzug, eine Namensänderung durch Heirat oder die in immer kürzeren Abständen wechselnden E-Mail-Adressen und Handynummern. Damit dem Administrator die Last der Einpflegearbeit abgenommen werden kann, dürfen Benutzer ihre persönlichen Daten selbst pflegen. Das Aktivitätsdiagramm links veranschaulicht den Ablauf: Nach der Änderung der Daten bekommt der Administrator eine Hinweis-Mail geschickt und kann eine entsprechende Änderungsseite aufrufen. Dabei kann er entscheiden, ob er die Daten genehmigt und automatisch in die Datenbank übernimmt, oder ob er die Änderungen verwerfen will, z.B. weil sie unsinnige Daten enthalten (Benutzern wird es zuweilen manchmal etwas langweilig). Sobald die Daten bestätigt wurden, kann der Benutzer seine Änderungen sehen. Technisch gesehen existiert eine Kopie der Tabelle Person (Person_Temp), in der die geänderten Daten bis zur Datenprüfung durch den Admin vorgehalten werden.

5.4.2 Buchungen

Abbildung 20 zeigt den Buchungsvorgang für See- und Binnenschiffe. Bevor überprüft wird, ob das ausgewählte Boot seegängig ist oder nicht, muss die Terminüberprüfung erfolgreich verlaufen sein und der Benutzer muss die entsprechende Segelberechtigung aufweisen können. Binnenschiffe weisen die Besonderheit auf, dass Buchungen nur maximal bis zwei Wochen im Voraus getätigt werden dürfen, während Seeschiffs-buchungen grundsätzlich erst den Status „reserviert“ bekommen. Erst nachdem die Reservierung durch den zuständigen Obmann bestätigt wurde, wird eine Reservierung zu einer Buchung.

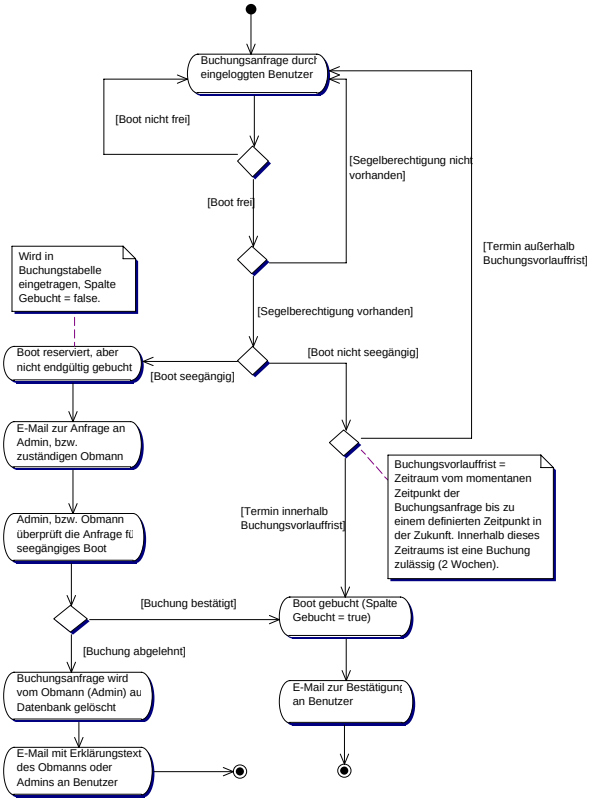


Abb. 20: Buchungsvorgang

Abbildung 21 zeigt den Vorgang einer Buchungsänderung. Buchungen, deren Startdatum bereits überschritten ist, können grundsätzlich nicht mehr vom Benutzer geändert werden. Auch hier unterscheiden sich die Binnenschiffe auf der Alster von den Seeschiffen auf Nord- und Ostsee: Die Buchung eines Binnenschiffs ist bis zum Startzeitpunkt fristgerecht änderbar, bzw. auch stornierbar. Bei Seeschiffen existiert dagegen eine Sperrfrist von derzeit acht Wochen vor Buchungsbeginn, in der nichts mehr fristgerecht geändert werden kann. Falls der Skipper, auf den die Buchung läuft, den Törn doch nicht antreten kann, so muss er ein Formular mit seiner Begründung ausfüllen, was der zuständige Obmann oder der Admin (falls für dieses Boot kein Obmann eingeteilt ist) per E-Mail zugeschickt bekommt. Der Obmann kann der Buchung nun den Status „virtuell“ zuweisen, was bedeutet, dass der Buchungszeitraum wieder als frei markiert wird um das Boot eventuell mit einer Ersatzcrew auszulasten. Die Buchung taucht aber trotzdem noch in der Abrechnung/Auflistung der Buchungen auf. Angenommen eine Ersatzcrew übernimmt den Törn, so muss der Obmann die Buchung auf den ursprünglichen Skipper löschen, andernfalls ist die virtuelle nicht angetretene Buchung trotzdem abrechenbar.

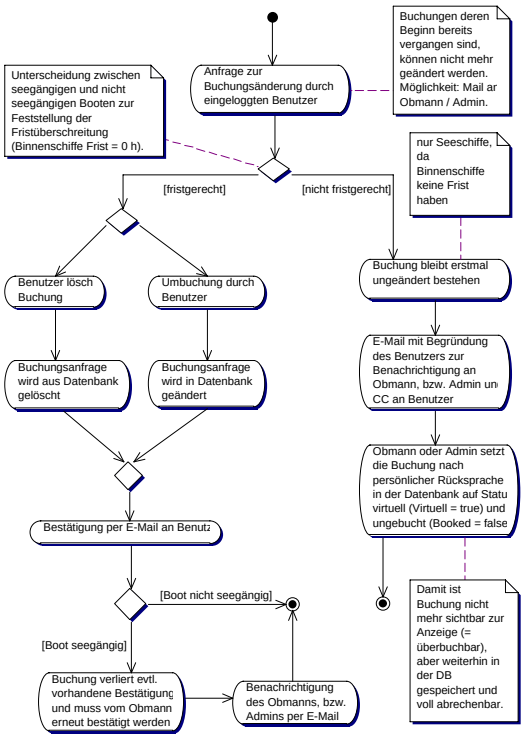


Abb. 21: Buchungsänderung durch User

5.5 Funktionsbeschreibung Obmannzugang

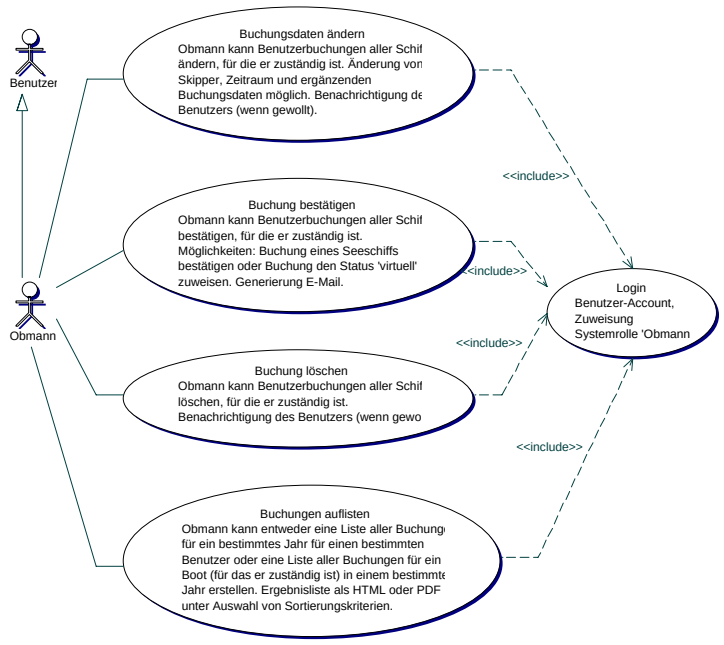


Abb. 22: Funktionsbeschreibung Obmannzugang

Der Obmann darf Buchungen von Schiffen, für die er verantwortlich ist, bearbeiten, bestätigen oder löschen. Zusätzlich kann er den Status „virtuell“ setzen oder aufheben. Bei den meisten Aktionen kann er entscheiden, ob und mit welchem Text der Skipper per Mail benachrichtigt wird. Die Funktion „Buchungen auflisten“ erstellt je nach Wunsch eine HTML-Seite oder ein PDF-Dokument zum Drucken mit der Auflistung aller Buchungen in einem auswählbaren Jahr. Dabei kann eine Auflistung entweder für einen Benutzer oder für ein vom Obmann betreutes Boot erstellt werden.

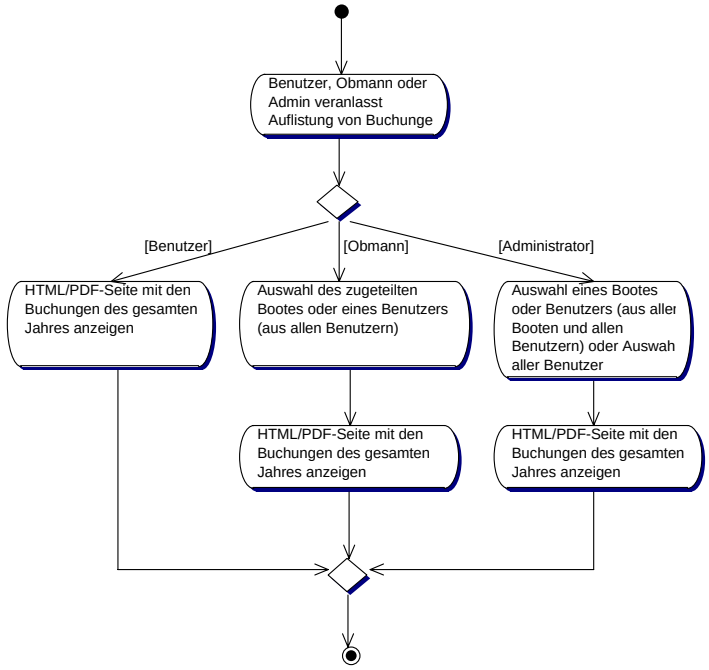
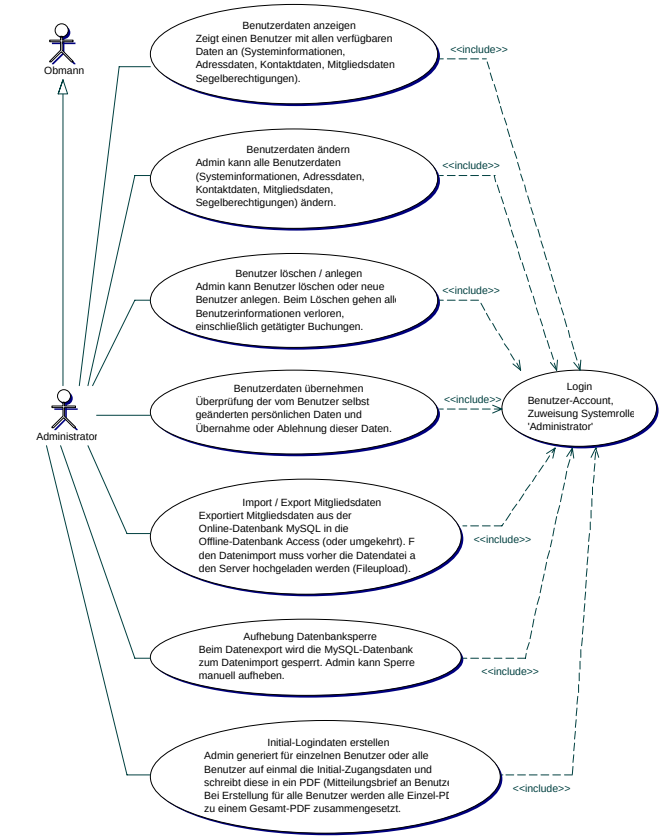


Abb. 23: Buchungsauflistung je nach Benutzerrechten

Das Activity-Diagramm links geht speziell auf die zuletzt genannte Funktion „Buchungen auflisten“ ein. Diese Funktion steht allen Benutzern vom Gast bis zum Administrator zur Verfügung, jedoch in unterschiedlicher Ausprägung. Die Unterschiede sind dem Schaubild links zu entnehmen. Gastbenutzer sind nicht erwähnt, da sie die Funktion nicht sinnvoll nutzen können (ist nur zu Demonstrationszwecken für eine Demo-Buchung aufrufbar). Alle Benutzer können im Modus „Auflistung für einen Benutzer“ auswählen, ob die Buchungen in zeitlicher Reihenfolge oder nach Booten gruppiert in zeitlicher Reihenfolge sortiert angezeigt werden sollen. Bei der Auflistung aller Buchungen für ein Boot macht die Gruppierung nach Booten natürlich keinen Sinn, wirkt sich bei Auswahl aber auch nicht negativ aus.

5.6 Funktionsbeschreibung Administratorzugang



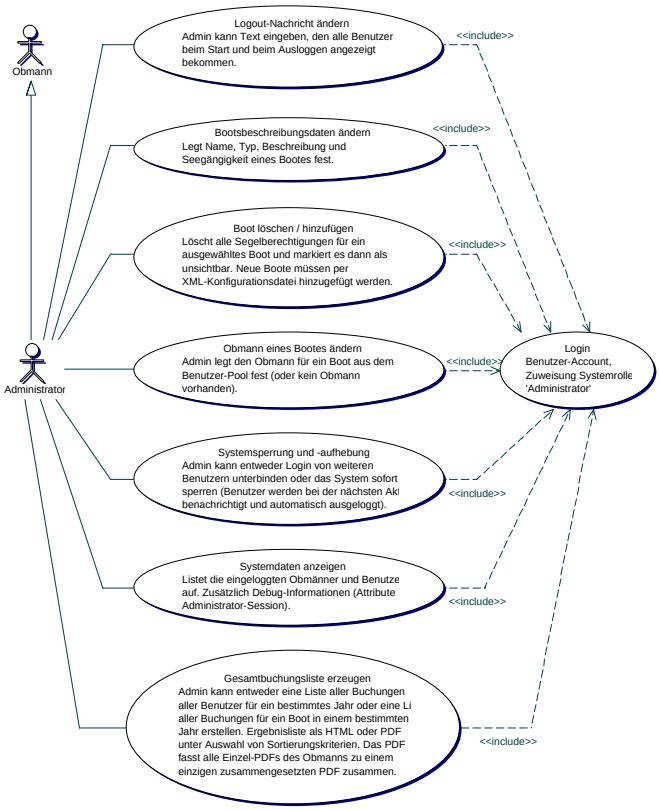


Abb. 24: Funktionsbeschreibung Administratorzugang

Die beiden Use-Case-Diagramme für den Administrator wurden zu einer einzigen Abbildung zusammengefasst. Neben der geerbten Möglichkeit Buchungsauflistungen für alle Benutzer und als Administrator auch für alle Boote zu erstellen, gibt es für den Admin die zusätzliche Möglichkeit, ein großes PDF zu generieren, das alle Einzel-PDFs zusammenfasst. Dies funktioniert sowohl bei allen Buchungsauflistungen für Benutzer wie auch für Boote. Selbstverständlich liefert auch die HTML-Ansicht alle Buchungen für alle Benutzer oder alle Schiffe. Das vereinfacht die Rechnungsstellung für jeden einzelnen Benutzer erheblich (ein Dokument anstatt 500 Dokumente ausdrucken).

5.6.1 Benutzer-Management

DiLog verfügt über ein integriertes komplettes Benutzer-Management: Benutzerdaten anzeigen und ändern, neue Benutzer anlegen und vorhandene Benutzer löschen. Dazu kommt die schon angesprochene Möglichkeit der Datenübernahme von durch den Benutzer selbst eingegebenen Änderungen. Zum Benutzer-Management gehört auch das Verwalten der Segelberechtigungen.
Hinweis: Wenn ein Benutzer gelöscht wird, so müssen aus Gründen der Datenintegrität neben seinen Segelberechtigungen etc. auch seine Buchungen gelöscht werden, egal ob bezahlt oder unbezahlt. Benutzer also erst löschen, wenn alle Rechnungen beglichen wurden!

5.6.2 Datenexport und -import

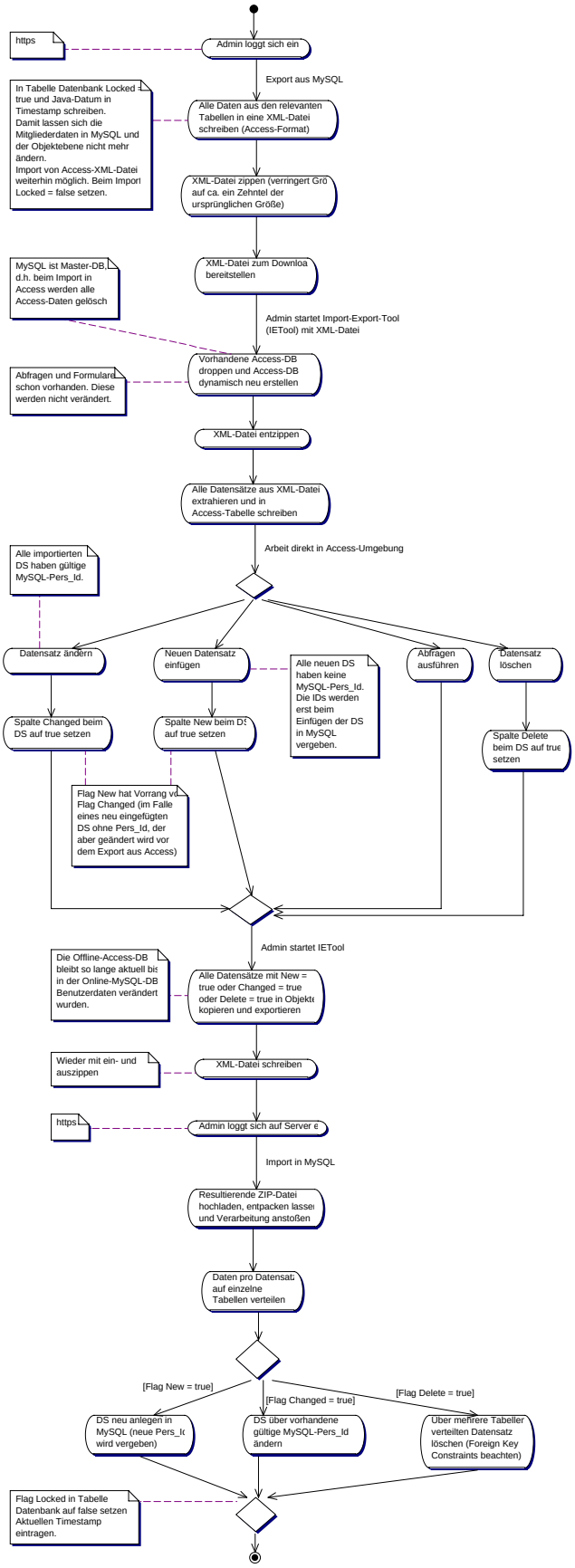


Abb. 25: Export-Import-Zyklus Mitgliederdaten

Das abgebildete Aktivitätsdiagramm stellt unter Verwendung des IETools und der Access-Datenbank den kompletten Zyklus vom Datenexport (aus MySQL heraus) bis zum Datenimport (nach MySQL zurück) dar. Anhand der Verzweigungen in der Mitte des Diagramms ist deutlich zu erkennen, wie die verschiedenen Flags in Access gesetzt werden und wie sie in der zweiten Verzweigung ganz am Ende beim MySQL-Import abgefragt werden.

5.6.3 Zugangsdaten

Der Administrator muss für jeden Benutzer zu Beginn einmalig einen Account anlegen. Dabei generiert DiLog ein sicheres Passwort und verknüpft es mit einer systemweit eindeutigen Benutzerkennung. Diese beiden Werte werden Initial-Logindaten genannt. Mit ihnen kann sich der Benutzer immer wieder anmelden, wenn er seine eigenen selbstgewählten Logindaten vergessen sollte. Die Initialkennung besteht aus <vorname><nachname><ID aus DB>. Das Initialpasswort ist eine vielstellige (unhandliche, aber ausreichend sichere) Kombination aus Buchstaben und Ziffern. Sie könnte beispielsweise so aussehen: SG0PSB6W3Z5O5.

Neben einem einzelnen Account können die Initial-Logindaten auch auf einen Rutsch für alle Benutzer erstellt werden. Es muss jedoch peinlichst genau darauf geachtet werden, dass noch keine Useraccounts bestehen, denn diese würden sonst überschrieben und der jeweilige betroffene Benutzer könnte sich nicht mehr einloggen. Nachdem der

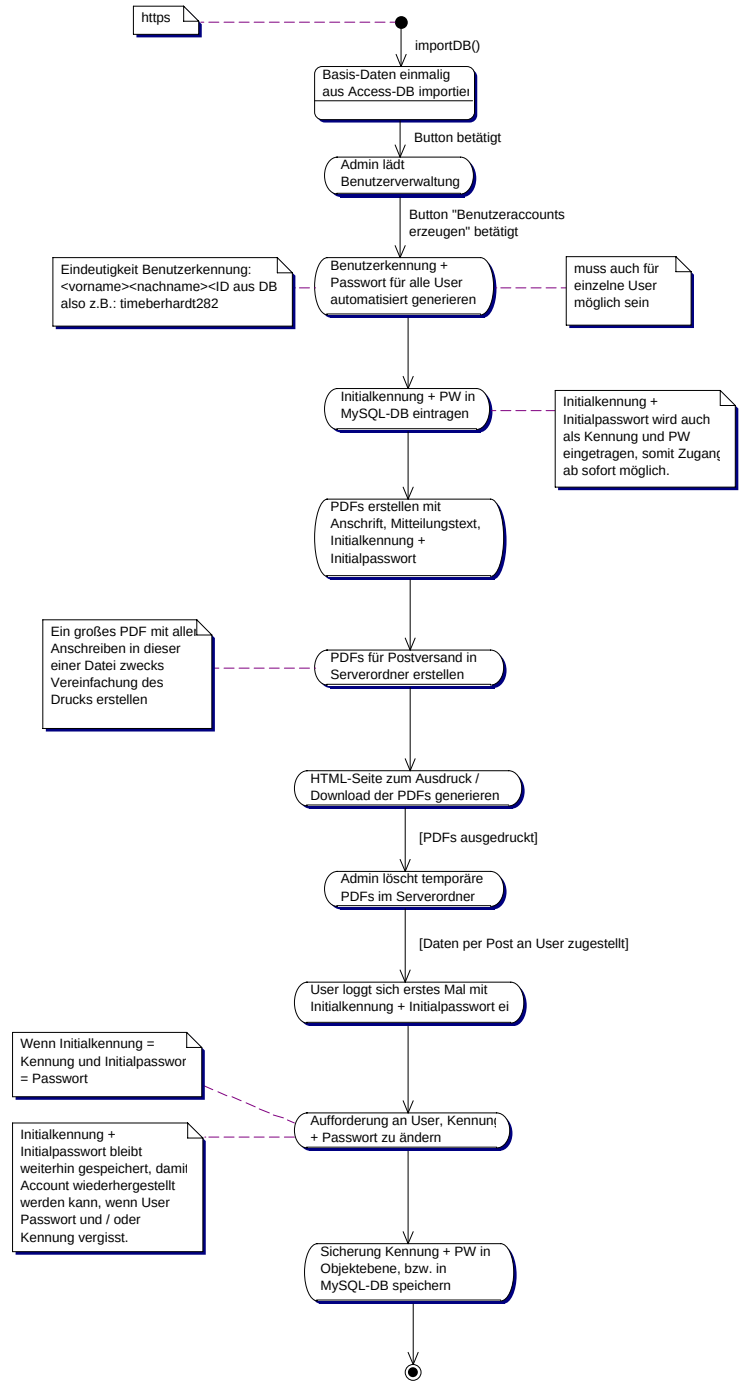
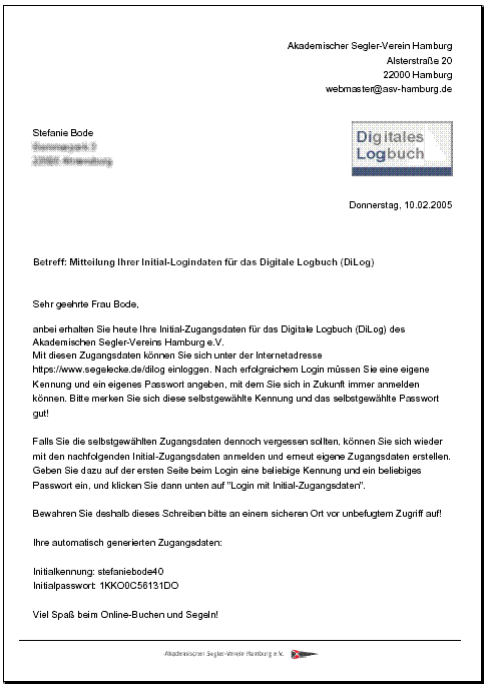


Abb. 26: Erstellung Initial-Logindaten

Zufallsgenerator das Passwort ausgeknobelt hat, wird ein PDF erzeugt, in das die Initial-Logindaten am Ende des Anschreibens eingefügt werden. Dieser Brief kann direkt einkuvertiert und postalisch an das entsprechende Mitglied verschickt werden. So sieht das mit der Java-Bibliothek iText generierte PDF aus:



Akademischer Segler-Verein Hamburg
Alsterstraße 20
22000 Hamburg
webmaster@asv-hamburg.de

Stefanie Bode
[redacted]

Digitales Logbuch

Donnerstag, 10.02.2005

Betreff: Mitteilung Ihrer Initial-Logindaten für das Digitale Logbuch (DiLog)

Sehr geehrte Frau Bode,

anbei erhalten Sie heute Ihre Initial-Zugangsdaten für das Digitale Logbuch (DiLog) des Akademischen Segler-Vereins Hamburg e.V.
Mit diesen Zugangsdaten können Sie sich unter der Internetadresse https://www.segelecke.de/dilog einloggen. Nach erfolgreichem Login müssen Sie eine eigene Kennung und ein eigenes Passwort angeben, mit dem Sie sich in Zukunft immer anmelden können. Bitte merken Sie sich diese selbstgewählte Kennung und das selbstgewählte Passwort gut!

Falls Sie die selbstgewählten Zugangsdaten dennoch vergessen sollten, können Sie sich wieder mit den nachfolgenden Initial-Zugangsdaten anmelden und erneut eigene Zugangsdaten erstellen. Geben Sie dazu auf der ersten Seite beim Login eine beliebige Kennung und ein beliebiges Passwort ein, und klicken Sie dann unten auf "Login mit Initial-Zugangsdaten".

Bewahren Sie deshalb dieses Schreiben bitte an einem sicheren Ort vor unbefugtem Zugriff auf!

Ihre automatisch generierten Zugangsdaten:

Initialkennung: stefaniebode40
Initialpasswort: 1KKO0C56131DO

Viel Spaß beim Online-Buchen und Segeln!

[redacted]

Abb. 27: Generiertes PDF

Das PDF kann über einen dynamisch erstellten Link von einem Temp-Verzeichnis des DiLog-Servers heruntergeladen werden. Natürlich ist es mit einem Passwort gesichert, das in der PDF-Konfigurationsdatei `/dilog/conf/clubaddress.properties` hinterlegt werden kann. Dort werden auch die Absenderadresse des Vereins und der Pfad auf den ASV-Stander (Vereinslogo in der Fußzeile) eingestellt.

5.6.4 System-Management

Der Administrator kann Bootsbeschreibungen (Name, Typ, Beschreibung, Seegängigkeit) ändern, Boote löschen oder hinzufügen, einem Boot einen Obmann aus dem Pool der User

zuweisen, die Datenbanksperre manuell aufheben, das System für Benutzer sperren (bzw. Sperre aufheben), Willkommens- und Abmeldenachricht einstellen und Systemdaten einsehen. Um konkurrierende Zugriffe zu verhindern ist nur ein Administrator gleichzeitig im System erlaubt.

Wenn dem System ein neues Boot hinzugefügt werden soll, muss es im ersten Schritt in die Konfigurationsdatei `boatconf.dilog.xml` eingetragen werden. Beim nächsten Systemstart ist das Boot dem System bekannt, und es können nun die Beschreibungsdaten eingegeben werden. Während alle Boote immer aus der XML-Datei gelesen werden, werden die Bootsbeschreibungsdaten immer aus der Datenbank gelesen.

Die Systemsperre soll noch kurz angeschnitten werden: Es gibt sie in zwei Varianten, und zwar in einer scharfen Form und in einer freundlicheren Art. Soll das System sofort alle Benutzer ausloggen, so wird die totale Systemsperre aktiviert. Jeder Benutzer, der gerade eingeloggt ist, bekommt bei seinem nächsten Seitenaufruf die Meldung angezeigt, dass das System ab jetzt für eine gewisse Zeit nicht zur Verfügung steht. Die freundlichere Variante ist die Login-Sperrung, was bedeutet, dass sich keine weiteren Benutzer mehr anmelden können, die bereits eingeloggten User aber nicht an der weiteren Ausführung von DiLog gehindert werden. Über die Systemdaten kann der Administrator erkennen, wer gerade (noch) eingeloggt ist und warten, bis sich alle Benutzer ausgeloggt haben.

5.7 Login-Prozesse

Wie bereits erwähnt, erhält der Benutzer zu Beginn seine Initial-Logindaten per Brief. Mit diesen Daten kann er sich einloggen, denn es gilt: Initialkennung = Kennung und Initialpasswort = Passwort. Wenn das System beim Login feststellt, dass diese Bedingung zutrifft, so wird der Benutzer aufgefordert, eine eigene Kennung und ein eigenes Passwort zu hinterlegen. Bricht er die Aktion ab, so wiederholt sich die Prozedur beim nächsten Einloggen.

Wenn sich der Benutzer beim Anmelden vertippt und sich damit mit falschen Daten anmeldet, bekommt er eine entsprechende Fehlermeldung präsentiert. Der Benutzer darf sich eine bestimmte Anzahl an fehlgeschlagenen Anmeldeversuchen leisten (derzeit 3 Versuche). Danach wird er zur Anmeldung mit den Initial-Logindaten aufgefordert. Bei einem fehlgeschlagenen Versuch wird die Session-Id in die Datenbank geschrieben und bei weiteren Fehlversuchen eine Zählvariable erhöht. So kann der Server für jeden Client mitzählen, wie oft er versucht hat, sich mit falschen Daten anzumelden. Klappt die Anmeldungen mit Initial-Logindaten, müssen danach wieder eigene Logindaten eingegeben werden, andernfalls kann man den Administrator per Formular-Mail beauftragen, neue Initial-Logindaten zu generieren.

Beim Login werden der erstellten Session ein paar Werte hinzugefügt, u.a. Login-Datum, Rolle und PersID des Benutzers. Des Weiteren wird ein `SessionTimeout`-Objekt erzeugt und der Session hinzugefügt. Verfällt die Session wegen einer Zeitüberschreitung

(Benutzerinaktivität), so wird die Methode `valueUnbound()` dieses Objektes aufgerufen. Hier können dann noch Daten gespeichert oder Ressourcen freigegeben werden.

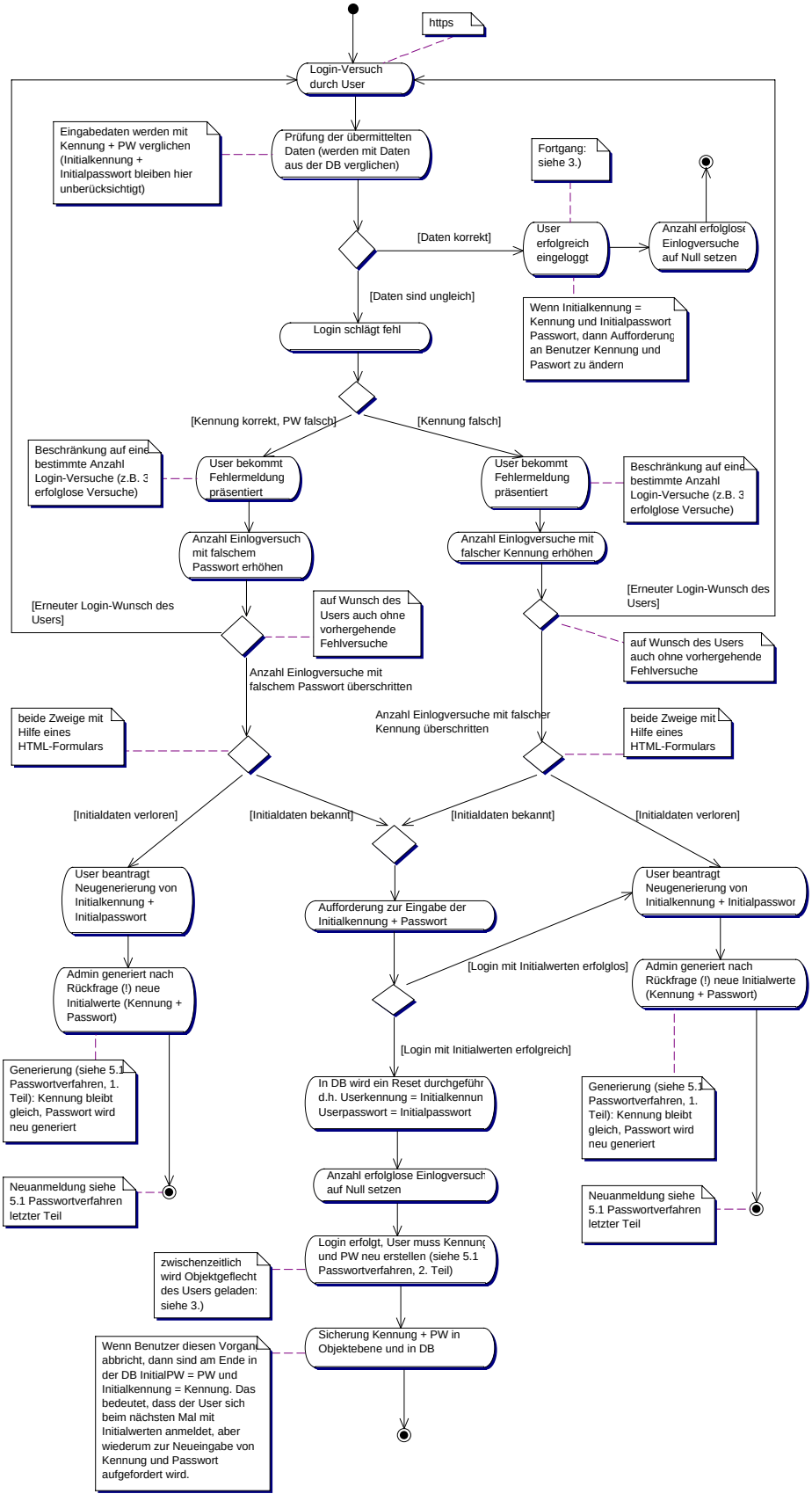


Abb. 28: Übersicht der Login-Abläufe in DiLog

5.8 Konfiguration

Es folgen nun noch ein paar Listings, bzw. Auszüge aus Listings und Hinweise zum Thema Konfiguration.

5.8.1 Datenbanken und DB-Manager

Die Access-DB muss als ODBC[52]-Datenquelle in den Windows-Systemeinstellungen angelegt werden. Dabei sollte unter System-DNS ein MS Access-Treiber hinzugefügt und die vorhandene Access-Datenbank ausgewählt werden.

Exkurs: Das Insert-Problem

Ein Problem speziell im Zusammenhang mit Access und JDBC beschäftigte mich mehrere Tage mit Debugging: Eine SQL-Insert-Anweisung wurde nicht in die Tabelle einer Test-Datenbank geschrieben, obwohl Java, bzw. JDBC als Rückgabewert 1 ausgab (= Anzahl der veränderten Zeilen). Die Symptome erinnerten stark an ein Puffer-Problem, denn das erste Insert wurde erst dann ausgeführt, wenn ein zweites Insert folgte, welches dann aber wiederum nicht ausgeführt wurde. Das Schließen der Verbindung führte ebenfalls das noch anstehende Insert aus. Ein manuelles `connection.commit()` half nicht weiter

Die Lösung ist eigentlich ganz einfach: In den erweiterten ODBC-Einstellungen muss das Flag „ImplicitCommitSync“ auf „Yes“ gesetzt werden. Dieser Konfigurationsfehler fiel deshalb nicht auf, weil bei Access 97 dieses Flag nach der Installation standardmäßig noch auf „Yes“ stand, bei Access 2000 jedoch „No“ vorgegeben ist.

Wenn man sich die zugehörige DTD anschaut, so ist klar, dass die DB-Konfigurationsdatei ein oder mehrere Tags `database` und einmal oder gar nicht das Tag `dbmanager` zulässt.
Während `dbconf.startdatatransformation.xml` für den einmaligen Transfer der Mitgliederdaten zwei Datenbanken definieren muss, kommt `dbconf.dilog.xml` mit einer Datenbank (MySQL) aus. Im Tag `database` müssen Angaben wie Treiber, Verbindungspfad und Authentifizierungsdaten gemacht werden, `dbmanager` ist für weitere Konfiguration zuständig, die sich auf den in DiLog implementierten DB-Manager auswirkt.

```
<database dbname = "MySQL">
   <driver>com.mysql.jdbc.Driver</driver>
   <path>jdbc:mysql://localhost:3306/Asv?autoReconnect=true</path>
   <!--path>jdbc:mysql://hightower500:3306/Asv?autoReconnect=true</path-->
   <!--path>jdbc:mysql://62.75.147.66:3306/Asv?autoReconnect=true</path-->
   <user>abc</user>
   <password>xyz</password>
</database>

<database dbname = "Access">
   <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
   <path>jdbc:odbc:ASV_V_2_1</path>
   <user></user>
   <password></password>
</database>
```

[52] ODBC (Open Database Connectivity) ist ein Datenbanktreiber, der es dem Entwickler über eine API ermöglicht, unabhängig vom verwendeten Datenbank-Server vorzugehen. Die Schnittstelle wurde ursprünglich für Access entwickelt, mittlerweile haben sie aber viele andere Programme übernommen.

5.8.2 Logging

Folgendes Listing wird für Log4J auf dem DiLog-Server im Debug-Modus verwendet:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

   <!-- FileAppender fuer den Standard-Logger -->
   <appender name="StandardAppender" class="org.apache.log4j.FileAppender">

      <param name="File" value="E:/server/tomcat5/webapps/dilog/logs/dilog.log"/>
      <param name="Append" value="false"/>

      <layout class="org.apache.log4j.PatternLayout">
         <param name="ConversionPattern" value="%d [%t] %p %c %x - %m%n"/>
      </layout>

      <filter class="magtime.helper.ChatLoggingFilter">
         <param name="stringToMatch" value="Chat" />
         <param name="acceptOnMatch" value="false" />
      </filter>

   </appender>

   <!-- ChatAppender fuer die Chat-Klassen -->
   <appender name="ChatAppender" class="org.apache.log4j.FileAppender">

      <param name="File" value="E:/server/tomcat5/webapps/dilog/logs/dilogchat.log"/>
      <param name="Append" value="false"/>

      <layout class="org.apache.log4j.PatternLayout">
         <param name="ConversionPattern" value="%d [%t] %p %c %x - %m%n"/>
      </layout>

   </appender>

   <!-- Logger fuer den Chat -->
   <logger name="Chat">
      <appender-ref ref="ChatAppender"/>
   </logger>

   <root>
      <priority value ="DEBUG" />
      <appender-ref ref="StandardAppender"/>
   </root>

</log4j:configuration>
```

Listing 14: Log4J-Konfigurationsdatei für DiLog

Es wird ein `FileAppender` als `StandardAppender` definiert. In ihn wird im Prinzip alles geloggt, was nicht für den `ChatAppender` vorgesehen ist. Dazu wird eine Datei angegeben und festgelegt, dass diese bei jedem Systemstart neu überschrieben werden soll. Als Layout wird das `ConversionPattern` verwendet. Mit dem Tag `filter` wird angegeben, dass alle Logaufrufe mit `Logger.getLogger("Chat");` ausgefiltert werden sollen, denn `stringToMatch` wird mit dem Wert „Chat“ belegt. Zusätzlich muss ein `Logger` und natürlich auch ein `FileAppender`, ebenfalls mit dem `ConversionPattern`, definiert werden.

5.8.3 Systemeinstellungen

In der Datei `systemconf.dilog.xml` werden alle das System an sich betreffenden Einstellungen getätigt. Dabei wird z.B. festgelegt, ob die beim Export von Mitgliederdaten erstellten XML-Dateien automatisch wieder gelöscht werden sollen, wie lange eine Session

dauern soll, Mail-Einstellungen werden hinterlegt, Einstellungen für den Upload der Webcam-Bilder, Chat-Einstellungen wie Portangabe und die Zeitangaben für Buchungsfristen. Die Datei ist gut kommentiert, deshalb hier keine weiteren Ausführungen.

5.8.4 Vereinsboote

Listing 12 zeigt eine in sich abgeschlossene Bootskonfiguration. Die Datei ist für DiLog natürlich etwas umfangreicher, da mehr Boote erfasst werden müssen. Das `equalBoat`-Tag wurde nötig, weil ein und dasselbe Boot unter mehreren Namen geführt wurde, meist als Standard-Segelerlaubnis und dann noch mit einer Einschränkung (meist Erlaubnis nur bis zu einer bestimmten Windstärke oder Jahreszeit). Damit Doppelbuchungen verhindert werden, muss dem System bekannt gemacht werden, dass es sich bei zwei verschiedenen Segelerlaubnissen um dasselbe Boot handelt. Das Tag `accessname` wird als Name der Tabellenspalte in der Access-Datenbank verwendet. Da Spalten nicht gleich heißen dürfen, sind unterschiedliche Namen unerlässlich. `mysqlname` wird als Bootsname in der MySQL-Tabelle Boote verwendet. `id` ist der Primary Key, d.h. alle vergebenen IDs müssen unterschiedlich sein. Einmal im System vergebene IDs dürfen nicht für andere oder neue Boote wiederverwendet werden. Genauer gesagt dürfen Boote gar nicht mehr aus der Datei entfernt werden, da alte Buchungen auf die Boote verweisen können (Datenintegrität!). Stattdessen wird dem Boot der Status „unsichtbar“ gegeben, was bedeutet, dass es im System nicht mehr angezeigt werden kann.

5.8.5 Sonstiges

Die Datei `applicationpath.properties` gibt den für die gesamte Applikation benötigten Pfad an. Er verweist auf das Oberverzeichnis von `conf`. Nötig wurde das, weil das Konfigurationsverzeichnis von Teilen der Applikation auch außerhalb Tomcat benötigt wird. So benutzen das IETool ebenso wie der Webcam-Client und die Applikation zur Datenübernahme aus dem Altsystem dieselbe Art der Konfiguration, nur mit unterschiedlichen Konfigurationsdateien.

```
# Der für die gesamte Applikation benötigte Pfad
# Oberverzeichnis von conf
# Pfadangabe ohne "/" am Ende
applicationpath=D:/da_asv_hh/eclipse
```

Listing 15: Festlegung Applikationspfad (Windows)

```
# Der für die gesamte Applikation benötigte Pfad
# Oberverzeichnis von conf
# Pfadangabe ohne "/" am Ende
applicationpath=usr/local/tomcat5/webapps/dilog
```

Listing 16: Festlegung Applikationspfad (Linux)

6 Technische Einführung in das Digitale Logbuch

Dieses Kapitel beschreibt hauptsächlich technische Aspekte von DiLog. Es sollen weniger Code-Beispiele gezeigt werden (bei der Fülle an Code müsste man ziemlich viel abdrucken), sondern vielmehr eine Übersicht auf die Struktur und die Abläufe. Dazu sind idealerweise Klassen- und Sequenzdiagramme geeignet. Doch auch hier gilt es, sich zu beschränken. Deshalb werden nur einige Teile beispielhaft beleuchtet, die sich aber ständig wiederholen und (nicht zuletzt dank Struts) übertragbar sind. Teilweise werden im vorigen Kapitel mit Ablaufdiagrammen umrissene Prozesse hier nun aufgegriffen und mit Sequenzdiagrammen technischer ausgelegt. Für den letzten Schritt – die Beschreibung der Vorgänge durch Code-Zeilen – wird auf den auf CD vorliegenden Source Code verwiesen. Das Eingehen auf konkreten Code würde den sowieso schon großen Umfang dieser Diplomarbeit sprengen.

6.1 Zusammenspiel von Tomcat und DiLog

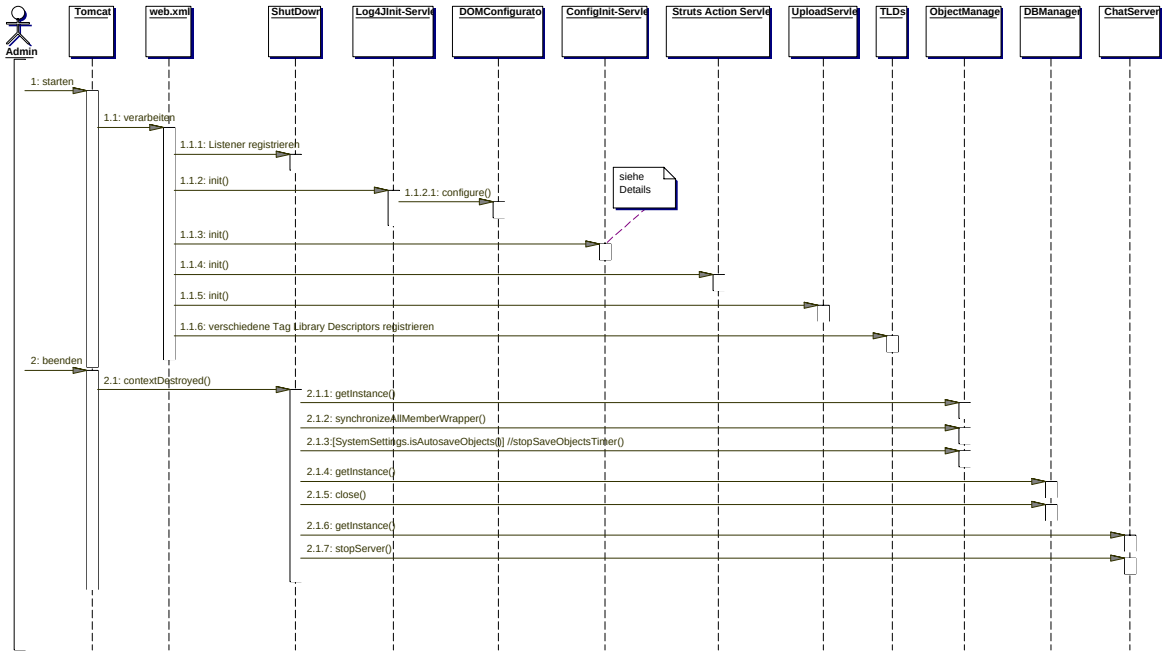


Abb. 29: Tomcat-Lebenszyklus und abhängige Objekte

Abbildung 29 spielt einen Lebenszyklus von Tomcat und seinen Objekten durch. Es wird aufgezeigt, was beim Beginn und beim Stoppen des Servers passiert. Beim Tomcat-Start wird die Datei `web.xml`, die im Anhang komplett abgedruckt ist, verarbeitet: Registrierung eines `ShutDown`-Listeners und Ausführung verschiedener Servlets. Die `ShutDown`-Klasse implementiert das Interface `ServletContextListener`. Ihre Methode `contextDestroyed()` wird aufgerufen, bevor beim Applicationserver das Licht ausgeht (oder wenn die DiLog-Applikation gestoppt wird). Was die Klasse `SessionTimeout` für die HTTP-Session ist, stellt `ShutDown` für den Server dar: Eine Möglichkeit um letzte Arbeiten vor der Zerstörung zu erledigen, hier die Speicherung von veränderten Daten aller im Moment eingeloggten Benutzer, das Schließen der Datenbankverbindung und das Beenden des Chat-Servers.

Noch bevor das Struts-Servlet das Framework initialisieren kann, wird Log4J über das `Log4JInit`-Servlet initialisiert, und über das `ConfigInit`-Servlet werden die verschiedenen Konfigurationsdateien gelesen. Nach Struts startet noch das `UploadServlet` um die Bilder des Webcam-Client in Empfang nehmen zu können. Außerdem werden die TLDs registriert. Das nächste Sequenzdiagramm ist ein Auszug aus dem vorhergehenden Diagramm, Grund des Herausnehmens ist die Darstellbarkeit auf Papier.[53]

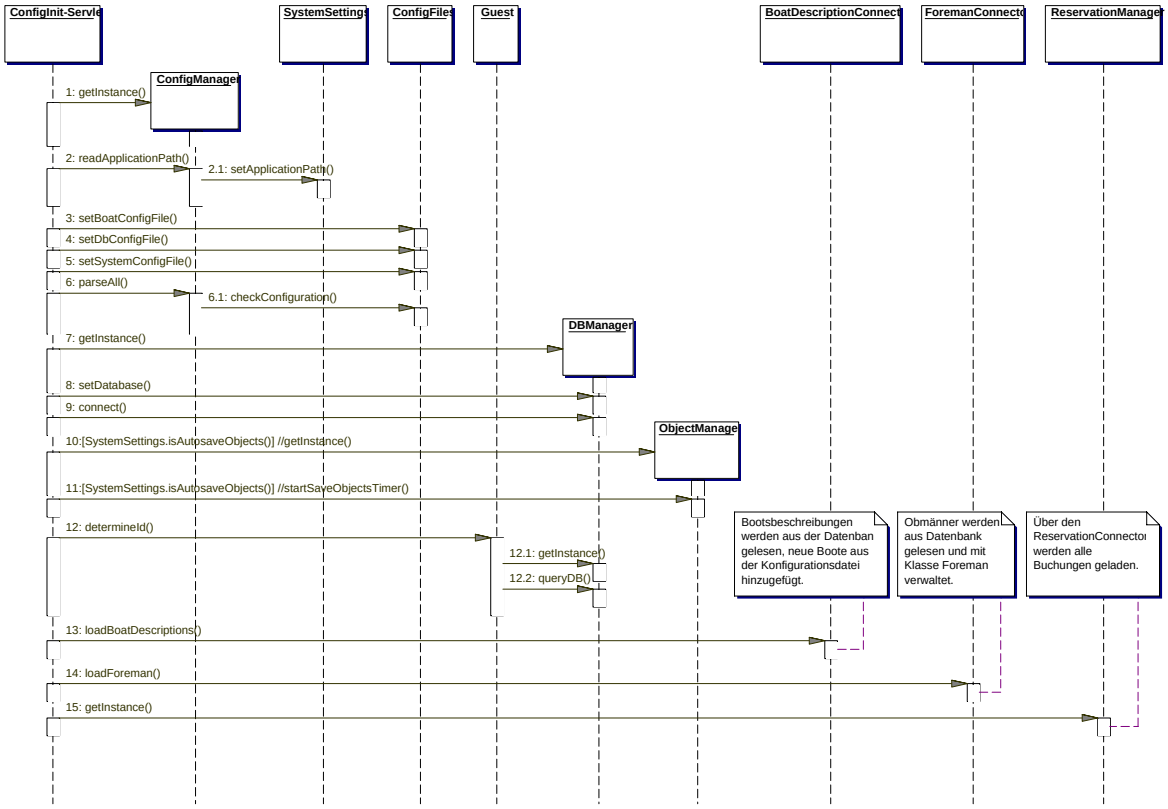


Abb. 30: Vorgänge beim Tomcat-Start (Servlet ConfigInit)

Die Aktionen oder Methoden 1-6 zeigen nochmals im Detail, wie der Konfigurationsvorgang abläuft. Danach wird über den DB-Manager die Datenbank gesetzt und eine Verbindung hergestellt. Der `ObjectManager` stellt für die sich einloggenden Benutzer die hinterlegten Daten bereit. Wenn er so konfiguriert ist, dass er in regelmäßigen Abständen Benutzerdaten sichern soll, so startet dieser extra Thread schon hier beim Start der Applikation. Prozess 12: determineId() ermittelt die PersId des Benutzers Gast. Das ist deshalb notwendig, weil dieser Benutzer beim Export von Mitgliederdaten herausgefiltert werden muss. Zusätzlich werden noch alle Bootsbeschreibungen, Obmann-Zuordnungen und Buchungen geladen.

6.2 Package Konfiguration

Zentraler Bestandteil des Konfigurationspakets ist der `ConfigManager`. Er liest alle Konfigurationsdateien ein und verwaltet deren Inhalt. Dazu muss er den in der Datei `applicationpath.properties` angegebenen Applikationspfad lesen (siehe Kapitel 5.8.5).

[53] Aus Platzmangel fehlen auch in allen UML-Sequenzdiagrammen die Linien, die andeuten, dass der Kontrollfluss zur aufrufenden Operation zurückkehrt.

Exkurs: Singleton-Prinzip

Das Singleton-Muster gehört zur Kategorie der sog. Creational Patterns innerhalb der bekannten Design Patterns. [Cooper 2000] (S. 39), dessen Tutorial in Anlehnung an das populäre Standardwerk „Design Patterns: Elements of Reusable Object-Oriented Software“ der „Gang of Four“ entstand, erklärt dazu:

> *This pattern is grouped with the other Creational patterns, although it is to some extent a pattern that limits, rather than promotes, the creation of classes. Specifically, it ensures that there is one and only one instance of a class and provides a global point of access to that instance.*

Die konkrete das Singleton betreffende Umsetzung sieht am Beispiel ConfigManager so aus:

```
package magtime.config.xml;
public class ConfigManager {

   private static ConfigManager configManager = null;

   private ConfigManager() {}

   public static synchronized ConfigManager getInstance() {

      if (configManager == null) {
         configManager = new ConfigManager();
      }
      return configManager;

   }

   // Methoden des Managers

}
```

Um die Instanz nun zu erlangen und um darauf eine Methode auszuführen geht man so vor:

```
ConfigManager cm = ConfigManager.getInstance();
cm.parseAll();
```

Alle Manager in DiLog (ConfigManager, ReservationManager, DBManager, ObjectManager) sind nach diesem Prinzip gebaut und innerhalb des ganzen Systems wie oben gezeigt aufrufbar.

Wie jeder DiLog-Manager ist auch `ConfigManager` als Singleton umgesetzt. Nachdem die benötigten Konfigurationsdateien angegeben wurden, folgt mit dem Aufruf `parseAll()` das Parsen der XML-Dateien. Dabei merkt sich `ConfigFiles` die einzelnen Dateinamen und meldet einen Fehler, wenn nicht alle Dateien gesetzt wurden. Wie im Klassendiagramm (Abbildung 31) ersichtlich ist, gibt es drei `XMLDocument`-Klassen (-Objekte), welche die zu parsenden Dateien repräsentieren: `BoatXMLDocument`, `DBXMLDocument` und `SystemXMLDocument`. Alle drei erben von derselben Basisklasse `XMLDocument`. Bei Erzeugung dieser Subklassen wird im Konstruktor dann das Parsen gestartet und später der erzeugte Dokumentenbaum vorgehalten.

Das Klassendiagramm zeigt auch die Methode `getData()`, die nachfolgend abgedruckt ist. Durch die Vererbung steht sie jeder `XMLDocument` erweiternden Klasse zur Verfügung. Der Methode wird ein XML-Knoten aus dem Dokumentenbaum übergeben, der dann gelesen werden soll.

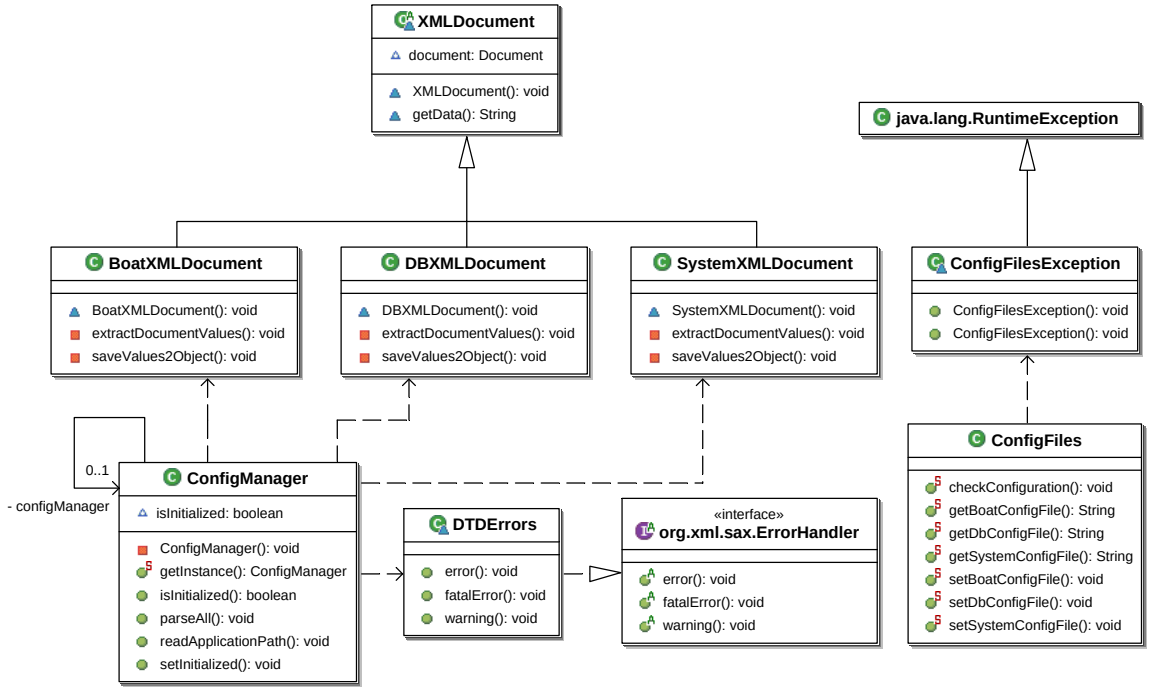


Abb. 31: Klassendiagramm Package magtime.config.xml

Aus einem XML-Knoten kann folgendermaßen ein Wert ausgelesen werden:

```
synchronized String getData(Node node) {

   if (node.hasChildNodes()) {
      return node.getFirstChild().getNodeValue();
   }
   return "";

}
```

Ein Textknoten speichert seinen Wert an der Stelle des „ersten Kindes“ ab. Bevor jedoch ein Wert aus einem Knoten gelesen werden kann, müssen die (richtigen) Knoten erst ausfindig gemacht werden. Denn ein XML-Baum besteht aus vielen Knoten, und nicht jeder ist von Interesse. Zudem gibt es auch andere Werte, z.B. Attribute, zu lesen. Deshalb werden die gewünschten Knoten „ausgesiebt“ und der gesuchte Wert mit Iterationen und bedingten Abfragen ermittelt.

```
private synchronized void extractDocumentValues() {

   NodeList boatList = document.getElementsByTagName("boat");

   int counter = 0;
   while (counter != boatList.getLength()) {

      Node boat = boatList.item(counter);

      if (boat.getNodeName().equals("boat")) {

         Element id = (Element) boatList.item(counter);
         String idS = (String) id.getAttribute("id");
         int boot_id = Integer.parseInt(idS);

         NodeList boatChilds = boat.getChildNodes();
         saveValues2Object(boatChilds, boot_id);
         counter++;
```

Listing 17: Auslesen von Bootsdaten aus der XML-Datei

```
      }

   }

}

private synchronized void saveValues2Object(NodeList nameList, int id) {

   for (int i = 0; i < nameList.getLength(); i++) {

      Node name = nameList.item(i);

      if (name.getNodeName().equals("mysqlname")) {
         BoatSettings.addMySQLBoatName(getData(name), id);
      }
      if (name.getNodeName().equals("accessname")) {
         BoatSettings.addAccessBoatName(getData(name), id);
      }
      if (name.getNodeName().equals("equalboat")) {
         BoatSettings.addEqualBoat(id, Integer.parseInt(getData(name)));
      }

   }

}
```

Listing 17: Auslesen von Bootsdaten aus der XML-Datei (Fortsetzung)

6.3 Datenbank-Package

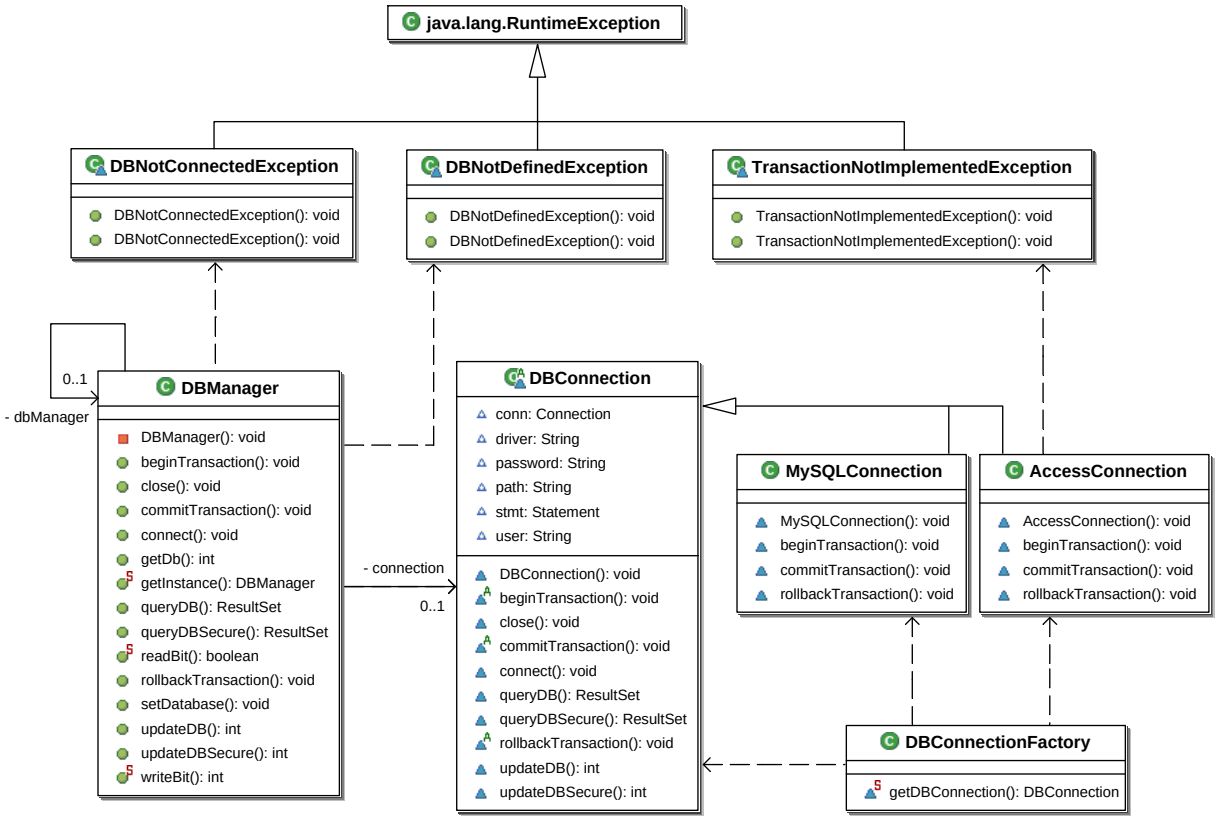


Abb. 32: Klassendiagramm Package magtime.db

Auch für dieses Package gibt es eine zentrale Anlaufstelle, den `DBManager`. Die abstrakte Klasse `DBConnection` stellt in Form einer abgeleiteten Klasse eine `MySQLConnection` oder eine `AccessConnection` bereit. Dabei entscheidet eine einfache Factory (`DBConnectionFactory`), welches Verbindungsobjekt konkret erzeugt werden soll.

6.3.1 Einmalige Datenübernahme aus Access

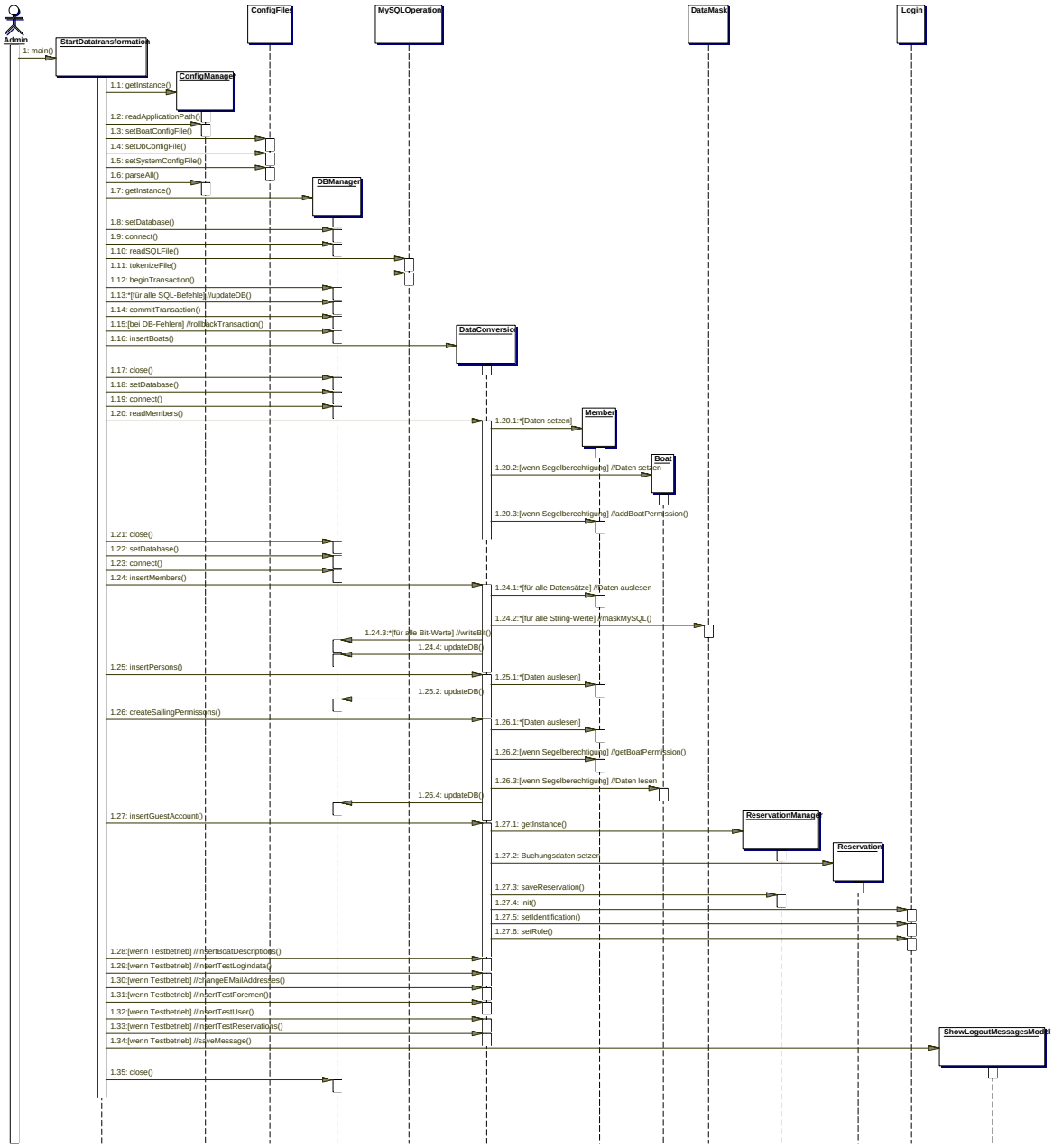


Abb. 33: Sequenzdiagramm Datenübernahme aus Access

Die einmalige Datenübernahme – eine Java-Applikation ohne Benutzeroberfläche – wird über die Klasse `magtime.start.StartDatatransformation` gestartet. Dabei müssen ihr zwei Parameter auf der Kommandozeile übergeben werden: innodb | myisam zur Angabe des Tabellentyps (InnoDB wird wegen Fremdschlüsseln und Transaktionen sehr empfohlen) und test | real, wobei im Testbetrieb zusätzlich einige zum Systemtest nötige Daten erzeugt werden.

Alle Access-Daten werden in die Objekte der Klasse `Member` eingelesen und nach Datenbankwechsel in die MySQL-DB geschrieben. Dabei entscheidet der Datenbankpfad in der Konfigurationsdatei `dbconf.startdatatransformation.xml`, welche MySQL-Datenbank angesprochen werden soll (Entwicklungssystem, Testsystem oder

Produktivsystem). Anschließend wird der Gast-Account angelegt. Im Testbetrieb werden zusätzlich alle E-Mail-Adressen umgeschrieben, Testlogindaten gesetzt, Testbuchungen angelegt und weitere Systemdaten in die Datenbank geschrieben.

6.3.2 Regelmäßiger Datenaustausch mit Access

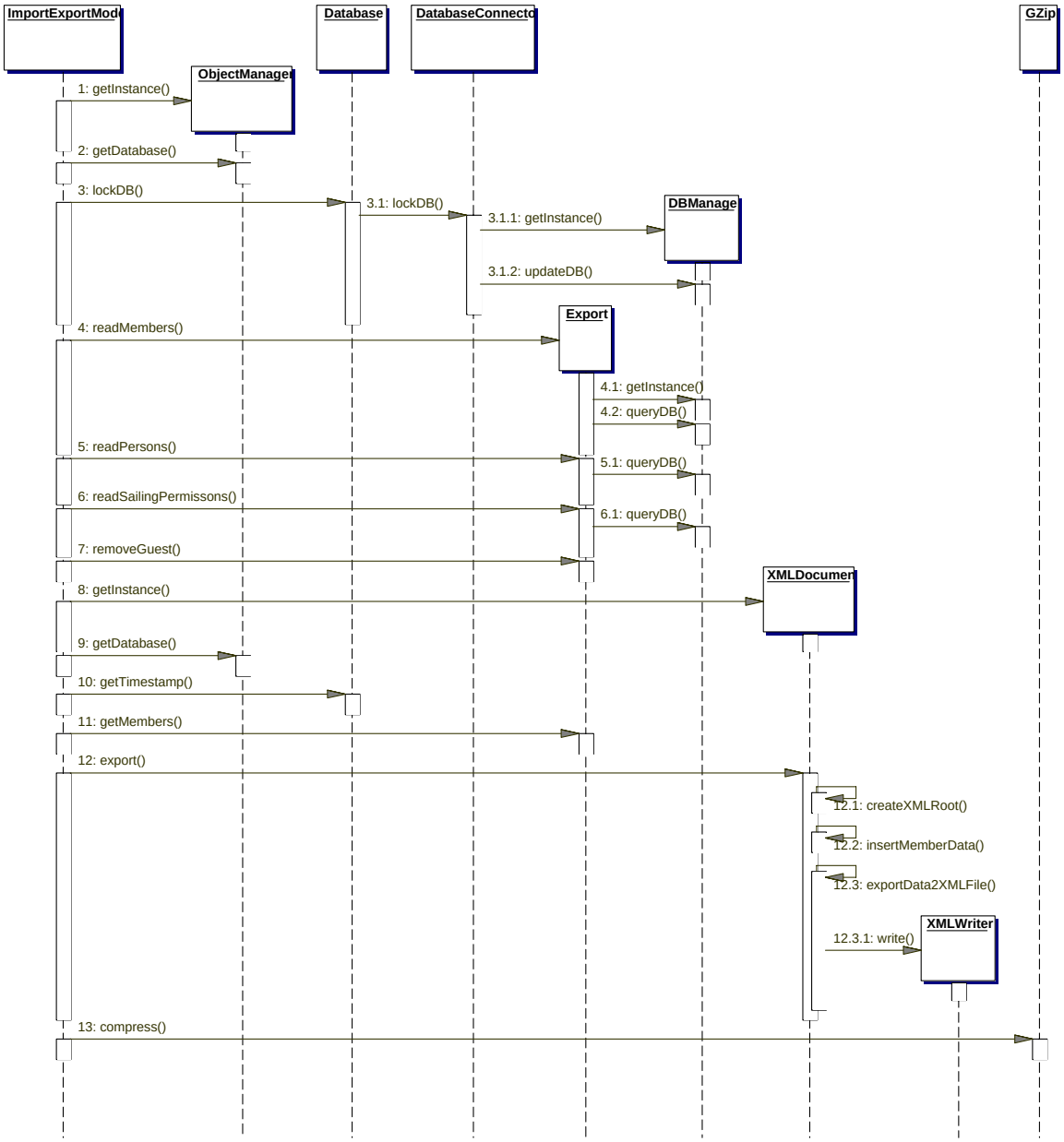


Abb. 34: Export von Mitgliederdaten durch Administrator

Es wurde gelegentlich schon geschildert, welchen Sinn der Export und Import von Mitgliederdaten hat. Deswegen folgt hier noch zur Komplettierung die abstrahierte Code-Ansicht in Form zweier Sequenzdiagramme. Abbildung 34 zeigt den Export der Daten aus MySQL, während die nachfolgende Abbildung den (Re-)Import der in Access geänderten Daten veranschaulicht. Auf die beiden jeweiligen Gegenstücke (Import nach Access und Export aus Access) geht Kapitel 7 ein.

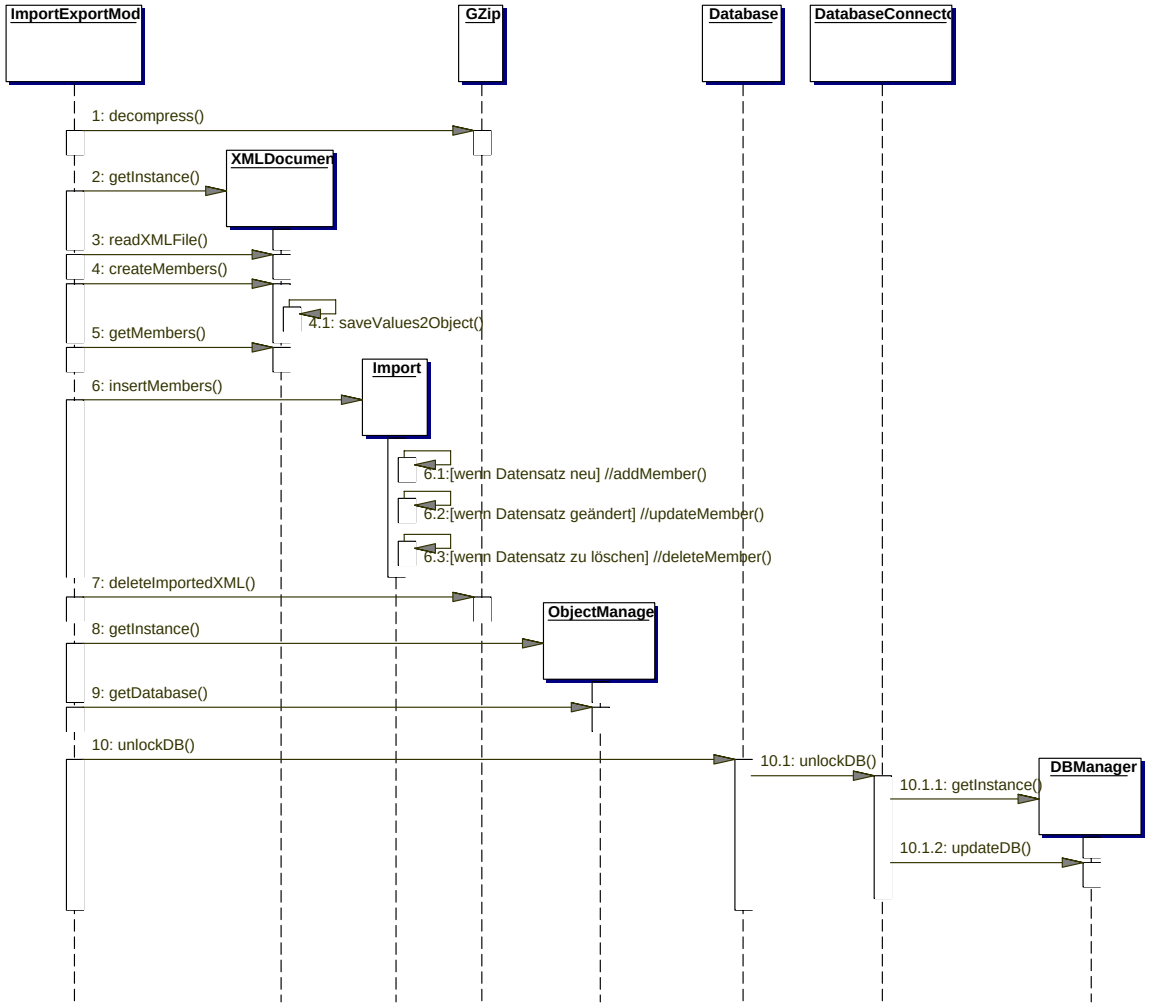


Abb. 35: Import von Mitgliederdaten durch Administrator

6.3.3 Connectors für Objektebene

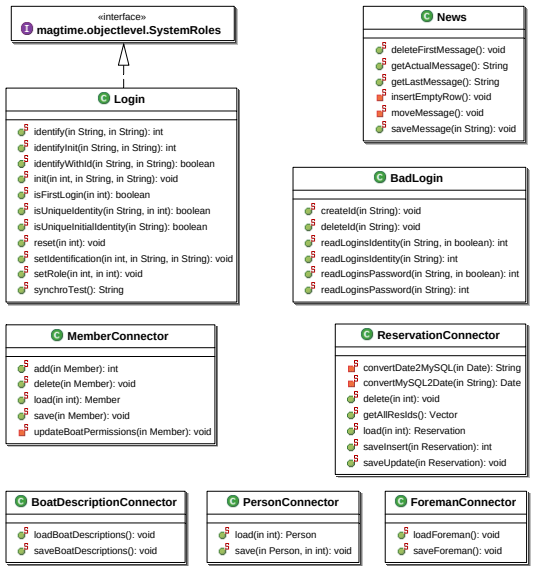


Abb. 36: Auswahl an Connector-Klassen

Connectors stellen in DiLog zusammen mit den Containern (Kap. 6.4.3) die Verbindung zwischen der Datenbankschicht und der Geschäftslogik dar (siehe Kapitel 6.4.1). In Abbildung 30 wurden Connectors bereits erwähnt: Das Start-Servlet `ConfigInit` weist den `BoatDescriptionConnector` mit `loadBoatDescriptions()` an, die Bootsdaten aus der Datenbank in den Datencontainer `BoatDescription` zu schaufeln. Nach demselben Prinzip lädt dort auch der `ForemanConnector` die Obmanndaten in den Container `Foreman`. Das Servlet erzeugt dann anschließend den `ReservationManager`, der im privaten

Konstruktor den `ReservationConnector` bemüht, alle vorhandenen Buchungen aus der Datenbank zu laden. `MemberConnector`, `PersonConnector`, `PersonTempConnector` und `LogindataConnector` werden durch die `MemberFactory` (Muster Fabrikmethode) zur Erzeugung eines `Member`-Objekts benutzt.

`Login` und `BadLogin` sind nicht direkt zu den Connectors zu zählen, da sie das Prinzip durchbrechen; Sie besitzen mehr Funktionalität als nur das Laden und Speichern von Daten. Zudem umgehen sie die Objektebene und operieren direkt auf den Daten in der Datenbank. Das wurde nötig, weil die Authentifizierung eines Benutzers direkt in der Datenbank mit Hilfe einer SQL-Anweisung stattfindet und beim Login zudem noch gar keine Benutzerdaten in die Objektebene geladen werden können, da ja vor der Authentifizierung noch gar nicht klar ist, um welchen Benutzer es sich überhaupt handelt.

6.4 Zwischenschicht Objektebene und Geschäftslogik

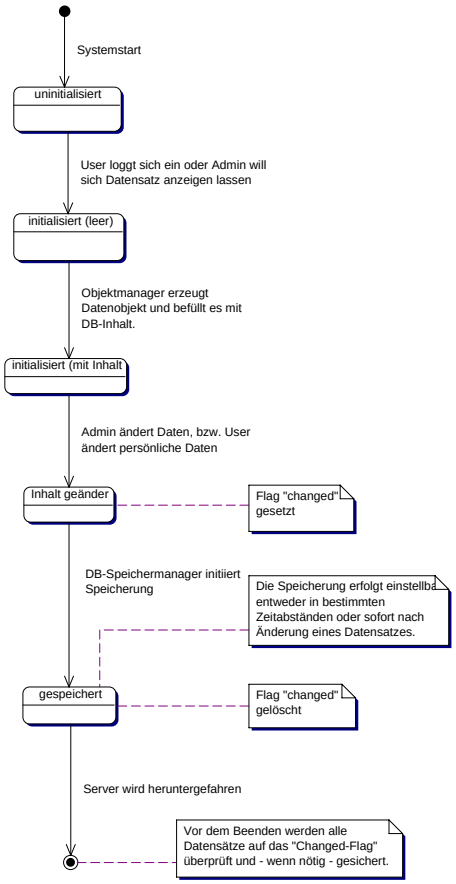


Abb. 37: Zustandsdiagramm eines Benutzerdatensatzes

Auf die in Kapitel 6.3 erläuterte Datenbankschicht setzt die Geschäftslogik mit der Objektebene auf. Zur Geschäftslogik können in erweiterter Form auch die Struts-Model-Klassen gezählt werden, denn auch sie setzen viele Geschäftsprozesse um. Genauer gesagt setzen die Struts-Model-Klassen auf die unterstützende Objektebene auf. Die Objektebene bietet die Basisfunktionalität, indem sie die Daten aus dem Backend aufbereitet und bereithält, auf welche die Model-Klassen dann bequem zugreifen können.

Zentraler Bestandteil der Objektebene ist wiederum ein Manager. Der `ObjectManager` kümmert sich um die einen Benutzer betreffende Datenhaltung und auch um die Änderung und Speicherung dieser Daten. Das UML-Zustandsdiagramm links demonstriert den Lebenszyklus eines solchen Benutzerdatensatzes. Auf den genauen Aufbau solch eines Datensatzes mit Containern geht Unterkapitel 6.4.3 ein.

6.4.1 DiLog-Schichtenmodell

Abbildung 38 zeigt das DiLog-Schichtenmodell im Servlet-Container von Tomcat. Nun werden die schon angesprochenen Verbindungen richtig deutlich. Die Datenbankschicht bildet das Backend und enthält die Datenbank und die Datenbankebene, die über den DB-Manager gesteuert wird. Darauf setzt die Geschäftslogik auf, gebildet von der Objektebene und dem Struts-Model. Der Objekt-Manager sorgt mit Hilfe von Containern und Connectors für den Datenfluss zwischen beiden Schichten. Ganz oben sitzt der restliche Teil des Struts-Frameworks mit seinen Actions und

Views. Der Datentransport zwischen der Geschäftslogik und dem Struts-Frontend (JSPs) wird durch ActionForm-Beans und Data-Beans gewährleistet.

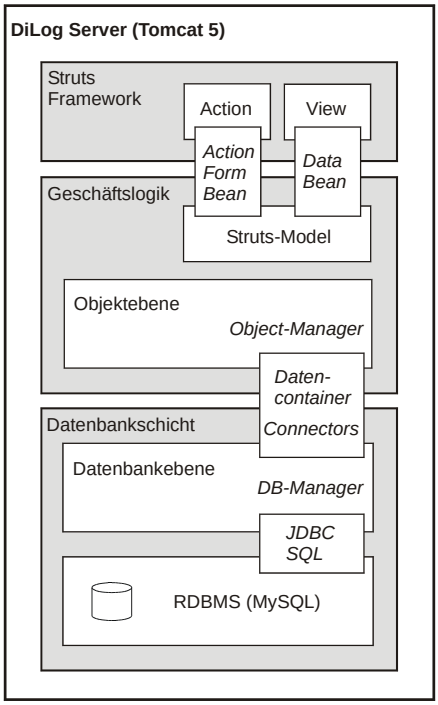


Abb. 38: DiLog-Schichtenmodell

Diese Benutzerdaten werden in einem im folgenden Unterkapitel erklärten sog. `MemberWrapper` zusammengefasst. Abbildung 39 verdeutlicht, wie solch ein `MemberWrapper` zur Bearbeitung durch den Administrator geladen und wieder entfernt wird (siehe auch Zustandsdiagramm Abb. 37). Dabei ist stets zu beachten, dass der entsprechende Benutzer während des Bearbeitungs-prozesses eingeloggt sein könnte.

6.4.2 Aufgabe der Objektebene

Die Objektebene versucht die Datenbankebene vor der Geschäftslogik zu kapseln. Dieses Verstecken des Backends hat zur Folge, dass es der Geschäftslogik plakativ gesprochen egal ist, woher die benötigten Daten kommen, Hauptsache sie sind vorhanden und nutzbar.

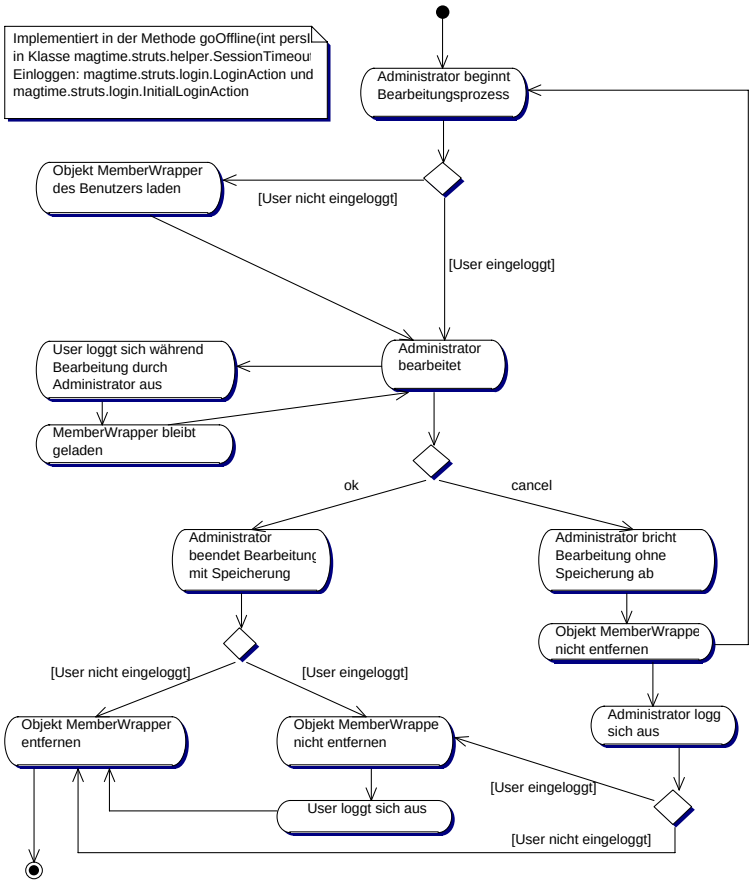


Abb. 39: Bearbeitung des MemberWrappers

6.4.3 Datencontainer

Ein `MemberWrapper` kapselt – wie der Name bereits andeutet – ein Objekt des Typs `Member`. `Member` selbst hat jeweils eine Referenz auf Objekte vom Typ `Person`, `PersonTemp`, und `Logindata`. Mit Ausnahme der Wrapper-Klasse erweitern sie allesamt die Klasse `DataContainer`. Die Container enthalten fast nur Getter und Setter und ein Paar Umwandlungsfunktionen für Datumsfelder. Der `MemberWrapper` hält keine direkten Mitgliedsdaten, sondern Verwaltungs- und Zustandsdaten wie Informationen darüber, ob er

neu angelegt wurde oder ob er in seinen zugeordneten Objekten geänderte Daten enthält. Das Klassendiagramm verdeutlicht die Bezüge untereinander noch einmal:

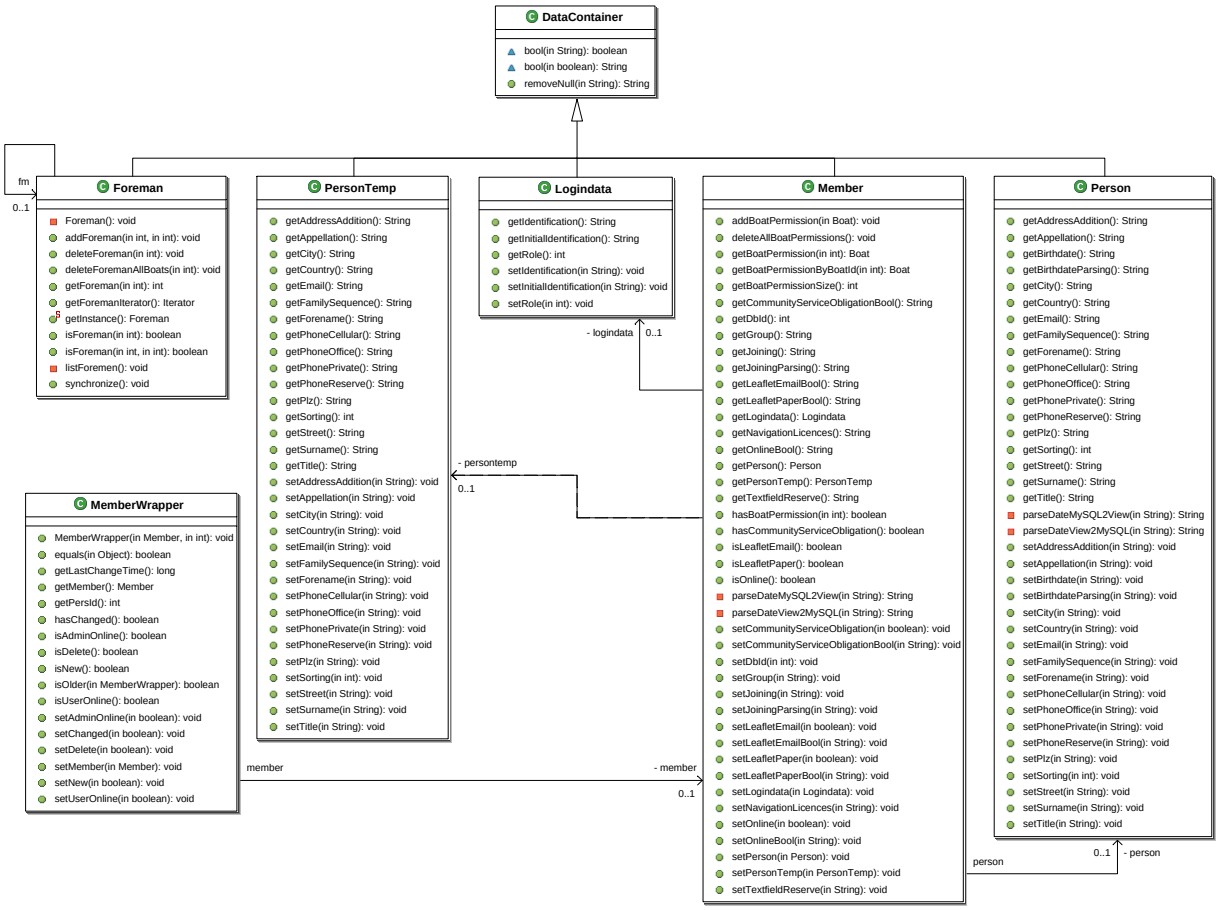


Abb. 40: Klassendiagramm Package magtime.objectlevel.datacontainer

6.5 Package Helper

Das Paket `magtime.helper` enthält einige nützliche Helferlein. Die Servlets `Log4JInit` und `ConfigInit` wurden bereits mehrfach erwähnt, ebenso wie der `ChatLoggingFilter` zur Unterdrückung der Chat-Logging-Daten. `ShutDown` erledigt Aufräumarbeiten, `Admin` und `PasswordThread` werden zum Erzeugen der Initial-Zugangsdaten benötigt. `ConvertTableNames` war für das Ant-Target `copydb` konzipiert, ist aber inzwischen nicht mehr notwendig, da keine MyISAM-Tabellen mehr benutzt werden. Zur Funktion und Aufgabe der Klasse steht in der JavaDoc-Dokumentation:

Diese Klasse ist notwendig, um beim Export der MySQL-Tabellen (die als Dateien existieren) von Windows auf Linux den ersten Buchstaben von Kleinschreibung auf Großschreibung zu ändern. Nach einem Underscore (_) muss ebenfalls ein Großbuchstabe folgen. Hintergrund: Unter Windows legt MySQL beim Erstellen einer Relation Testname die Tabelle testname an. Unter Windows kann man darauf mit Testname zugreifen, Linux findet die Tabelle jedoch nicht, da Linux-Systeme auf Groß- und Kleinschreibung in Dateinamen und Pfaden unterscheiden.

6.6 Struts-Package

Im Paket `magtime.struts` sind einige allgemeine Klassen abgelegt, die sich keinem der in den Unterkapiteln folgenden speziellen Paketen zuordnen lassen. Abbildung 41 zeigt die Vererbungshierarchie für alle verwendeten Form-Klassen: `BaseActionForm` definiert einige nützliche Zusatzmethoden und ist von der Struts-Klasse `ActionForm` abgeleitet. Alle `Action`-Klassen erben natürlich von der `Action`-Klasse im Struts-Framework. Wenn bestimmte Klassen den Benutzerstatus eines Users abfragen müssen, so implementieren sie dazu das Interface `SystemRoles`. Das Interface `NavView` enthält die Zuordnung eines `int`-Wertes zu einer `String`-Konstanten. Durch die Verwendung dieser Konstanten im JSP-Code des Navigationsmenüs können die `int`-Werte auch nachträglich noch geändert werden, falls dies durch irgendeinen Umstand nötig werden sollte. Zudem macht es den Code lesbarer und schützt vor Tippfehlern.

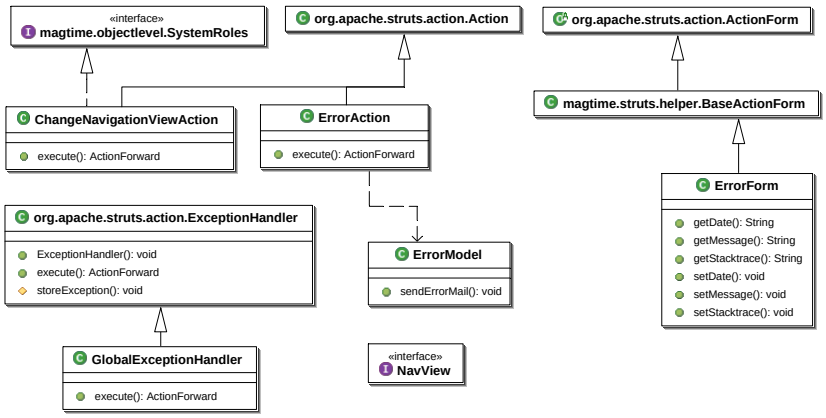


Abb. 41: Klassendiagramm Package magtime.struts

Die Klassen `ErrorAction`, `ErrorForm` und `ErrorModel` sind für das Error Reporting zuständig. Taucht in der Applikation ein Fehler auf, so bekommt der Benutzer die Möglichkeit, dem Administrator ein Formular per Mail zukommen zu lassen. Dabei kann er die Vorgänge beschreiben, die zu dem Fehler führten. In der E-Mail wird zusätzlich mit der Exception der Stacktrace verschickt.

6.6.1 Login-Package

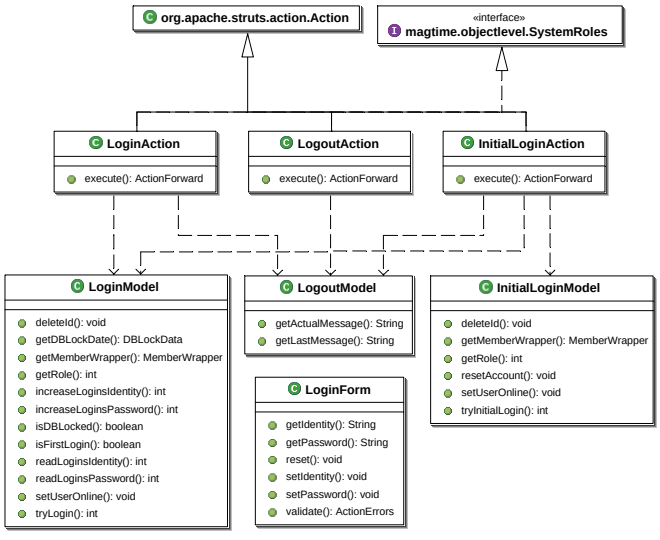


Abb. 42: Login-Package (1)

Das Login-Package ist ein umfangreiches und wichtiges Paket. Deshalb werden hier zwei Klassendiagramme und ein zusätzliches Sequenzdiagramm mit dem Vorgang des Einloggens eines Benutzers gezeigt.

Gut zu erkennen ist in Abbildung 42 die Tatsache, dass eine Action auf mehrere Model-Klassen zurückgreift. Dies ist deshalb der Fall, weil es nicht zweckmäßig ist, denselben Code mehrfach zu implementieren. Bei Änderungen in Model-Klassen muss dann allerdings genau überprüft werden,

ob die zu ändernde Methode evtl. noch von weiteren Actions (oder auch anderen Model-Klassen) verwendet wird. Wenn diese Abhängigkeiten nicht peinlichst genau beachtet werden, kann das zu bösen und schwierig zu findenden Fehlern führen. Das ist der Nachteil an der Mehrfachverwendung von Code.

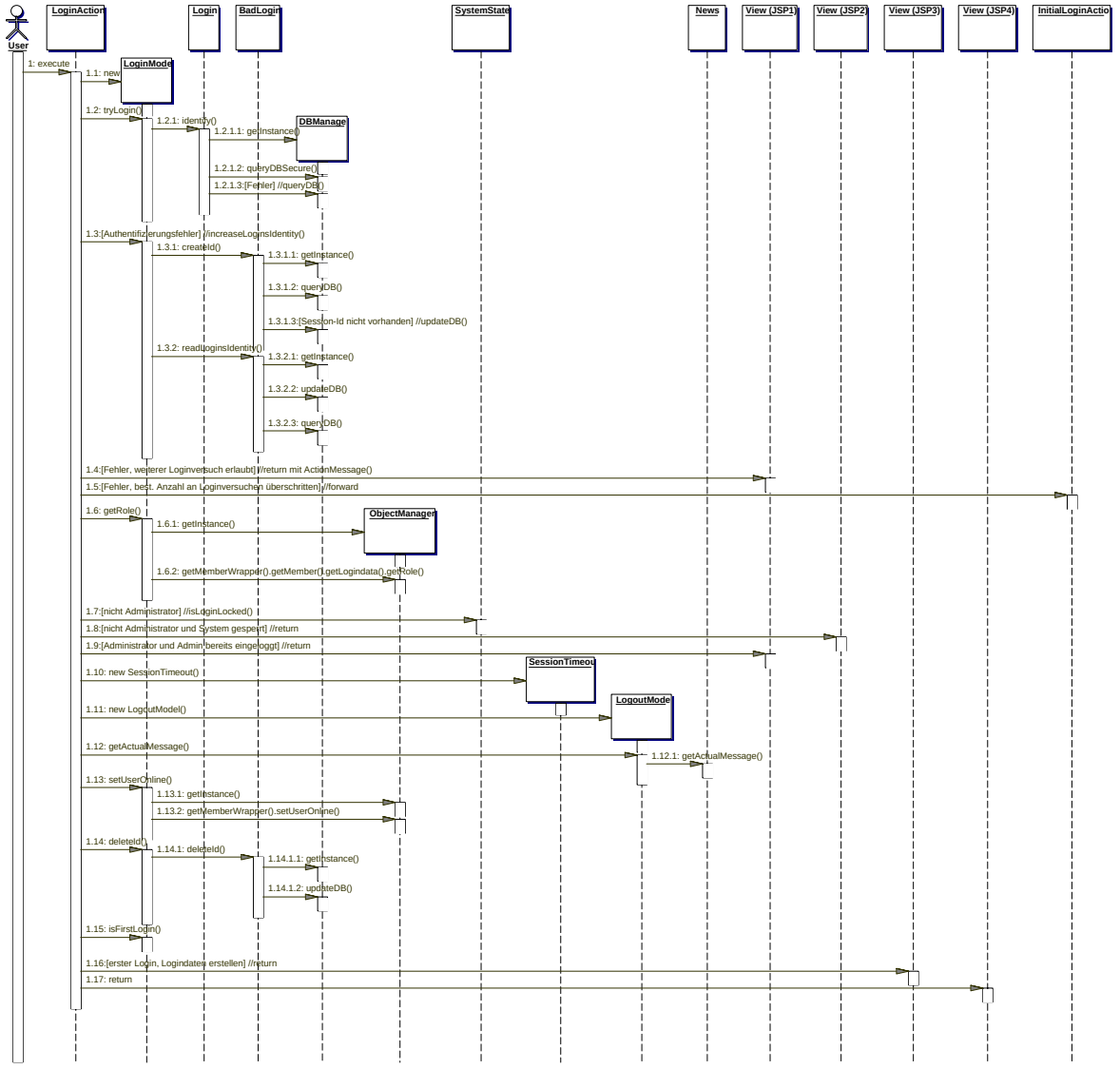


Abb. 43: Login-Vorgang mit allen Möglichkeiten des Programmflusses

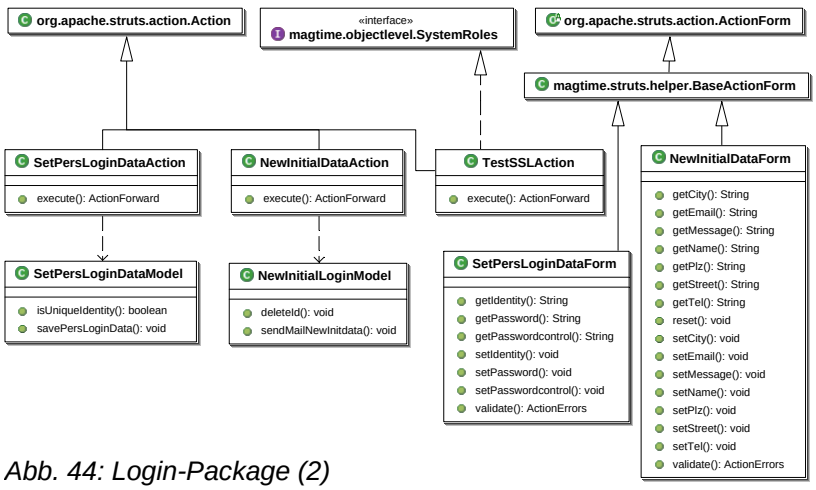


Abb. 44: Login-Package (2)

Das Sequenzdiagramm spielt alle Möglichkeiten durch, die auftreten können. Die Verwandtschaft zu Abb. 28 (Kapitel 5.7, Login-Prozesse) ist unübersehbar. Das Sequenzdiagramm drückt eher die Sicht auf den Source Code aus, während das Aktivitätsdiagramm den Schwerpunkt auf die reinen Abläufe legt.

6.6.2 Gast-Package

Es folgt zu jedem Benutzerstatus eine Darstellung eines ausgewählten Vorgangs. Schon allein die Masse der Diagramme legt nahe, dass die Vorgänge mit zunehmenden Rechten auch komplizierter und umfangreicher werden.

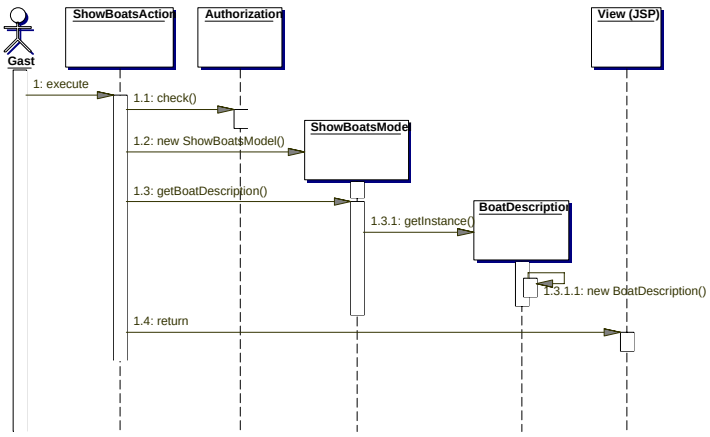


Abb. 45: Ablauf Vereinsboote anzeigen

Für den Status „Gast“ wurde eine Aktion ausgewählt, die einem Gast sämtliche Vereinsboote mit ihren Bootsbeschreibungen anzeigt. Die Klasse `Authorization` überprüft hier natürlich auch nur auf den Status „Gast“. Der Datencontainer `BoatDescription` selbst wird hier als Session-Bean an die View geschickt. Dies ist eine der wenigen Ausnahmen, wo ein in der Geschäftslogik verankertes Objekt an die JSPs ausgeliefert wird.

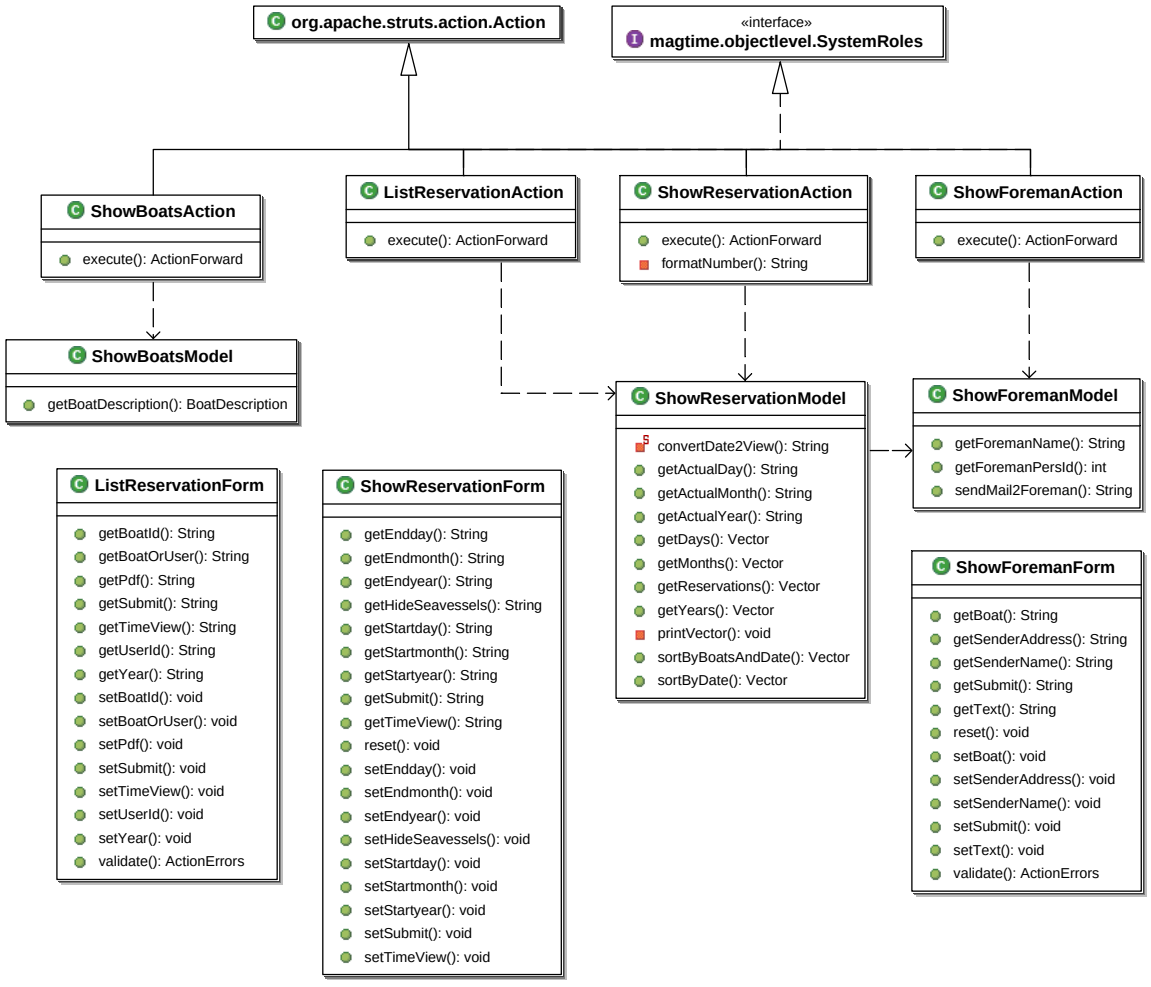


Abb. 46: Klassendiagramm Package magtime.struts.guest (Auszug)

6.6.3 Benutzer-Package

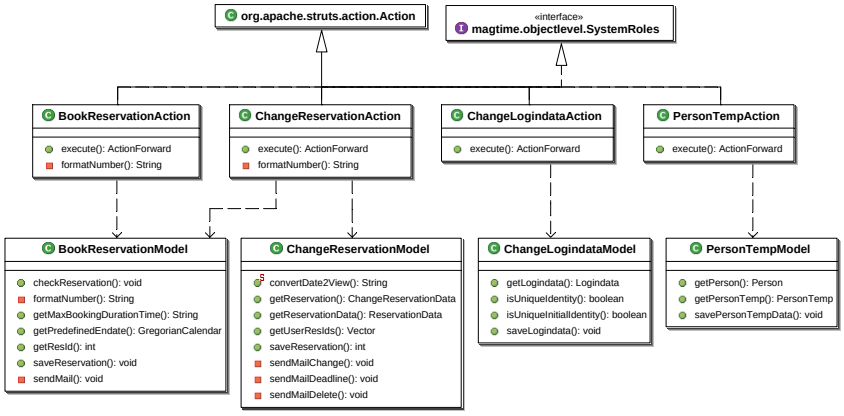


Abb. 47: Klassendiagramm Package magtime.struts.user

Dieses Beispiel zeigt, wie ein User seine Logindaten verändern kann. Dabei ist zu beachten, dass die Struts-Action (im Gegensatz zu der in Abb. 45 dargestellten Situation) zweimal aufgerufen wird, denn zuerst müssen die aktuellen Logindaten besorgt und angezeigt werden. Im zweiten Schritt werden die neuen Daten dann zurück gespeichert.

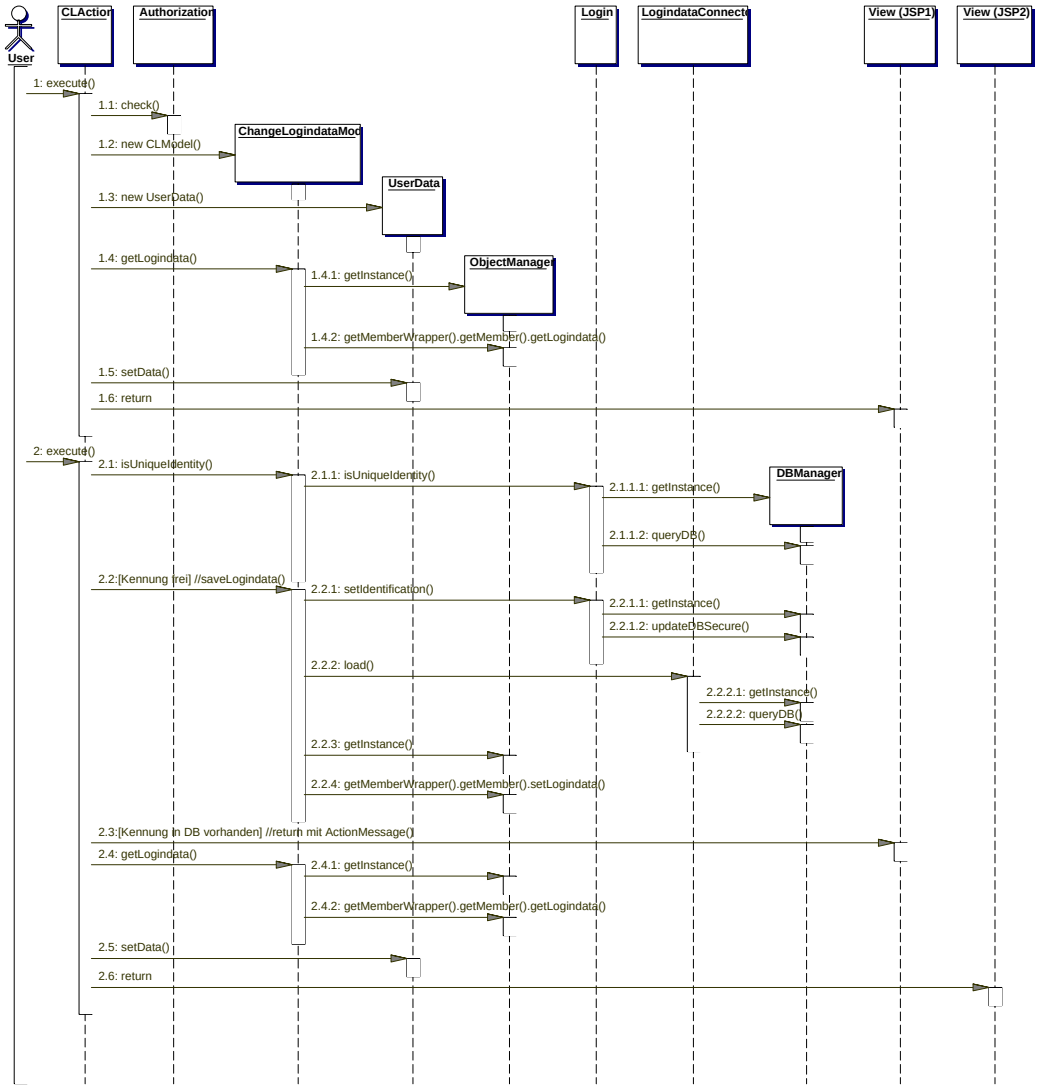


Abb. 48: Sequenzdiagramm zur Änderung der Logindaten durch einen Benutzer

6.6.4 Obmann-Package

Das Beispiel für das Obmann-Package ist ein ziemlich komplizierter und umfangreicher Geschäftsvorfall. Das zeigt sich schon daran, dass vier Sequenzdiagramme notwendig sind, um ihn komplett abzubilden. Er lässt sich grob unterteilen in ein Sequenzdiagramm für die Darstellung der Struts-Komponenten und drei Diagramme für die Methoden der Objektebene und zusätzliche Pakete wie Mail-Versand.

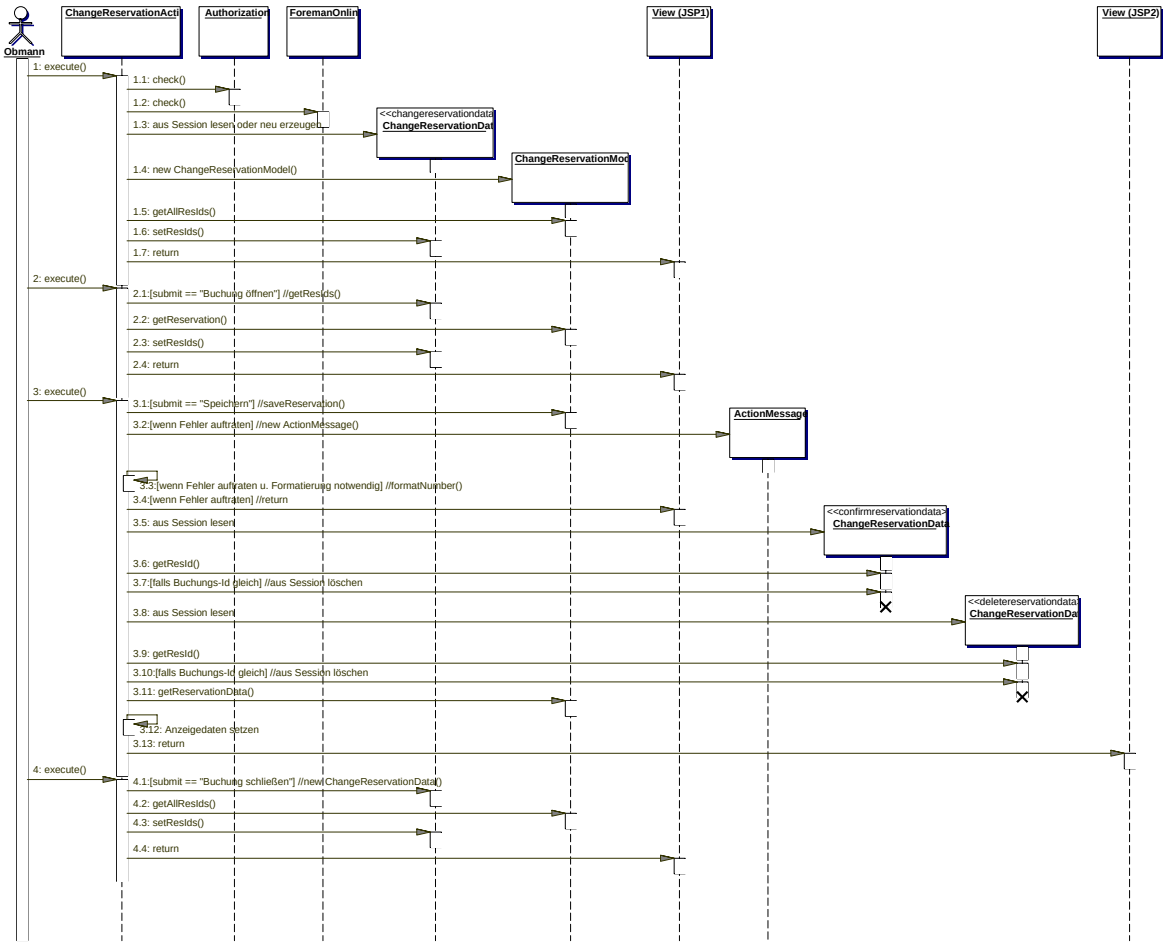


Abb. 49: Sequenzdiagramm Buchungsänderung durch Obmann (Struts-Ansicht)

Der ausgesuchte Prozess ist die Änderung einer Buchung durch den Obmann. Hier kann die Action bis zu viermal mit unterschiedlichen Vorzeichen aufgerufen werden (Aktionen 1-4). Es hätten auch vier unterschiedliche Actions ausprogrammiert werden können, jedoch würde das schnell zu einer unüberschaubaren Anzahl an Klassen führen. Statt dessen erkennt die `ChangeReservationAction` anhand des für den Submit-Button in der Variablen `submit` übertragenen Wertes, welche Methode der zugehörigen Model-Klasse gerade gewünscht ist.

Im ersten Schritt werden die Buchungsnummern der Buchungen ermittelt, die der Obmann bearbeiten darf. Wenn der Administrator diese Aktion ausführt, werden ihm alle Buchungen angezeigt. Der Obmann bekommt dagegen nur eine Untermenge aller Buchungen zu Gesicht, nämlich genau die Buchungen, die auf das Boot zutreffen, für das er zuständig ist. Die Buchungs-IDs werden als `Vector` im Bean `ChangeReservationData` gespeichert und

über die Session an die View ausgeliefert. Die JSP setzt die Daten dann mit folgenden Struts-Tags in eine Combobox:

```
<!-- zu ändernde Buchung noch nicht selektiert -->
<logic:notPresent name="changereservationdata" property="resId">
   <html:select property="resId" size="1">
      <html:optionsCollection name="changereservationdata" property="resIds" />
   </html:select>
</logic:notPresent>

<!-- zu ändernde Buchung bereits selektiert -->
<logic:present name="changereservationdata" property="resId">
   <html:select property="resId" value="<%= resId %>" size="1">
      <html:optionsCollection name="changereservationdata" property="resIds" />
   </html:select>
</logic:present>
```

Falls die Buchung noch nicht selektiert wurde, werden alle IDs in die Combobox eingetragen und die erste angezeigt. Im zweiten Fall wurde bereits eine ID ausgewählt, die als `resId` in das Bean mit dem Namen `changereservationdata` eingetragen wurde. Das Attribut `value` gibt die zuvor aus der Session gelesene und als `String`-Variable hinterlegte ID an. Das Tag `<html:optionsCollection />` trägt jeweils alle verfügbaren IDs aus dem Bean `ChangeReservationData` in die Combobox ein, wobei im zweiten Fall eben die zuvor schon einmal markierte ID selektiert angezeigt wird. Bei Übertragung des Formulars wird die momentan selektierte ID in der Variablen `resId` (Wert des Property-Attributs) übermittelt.

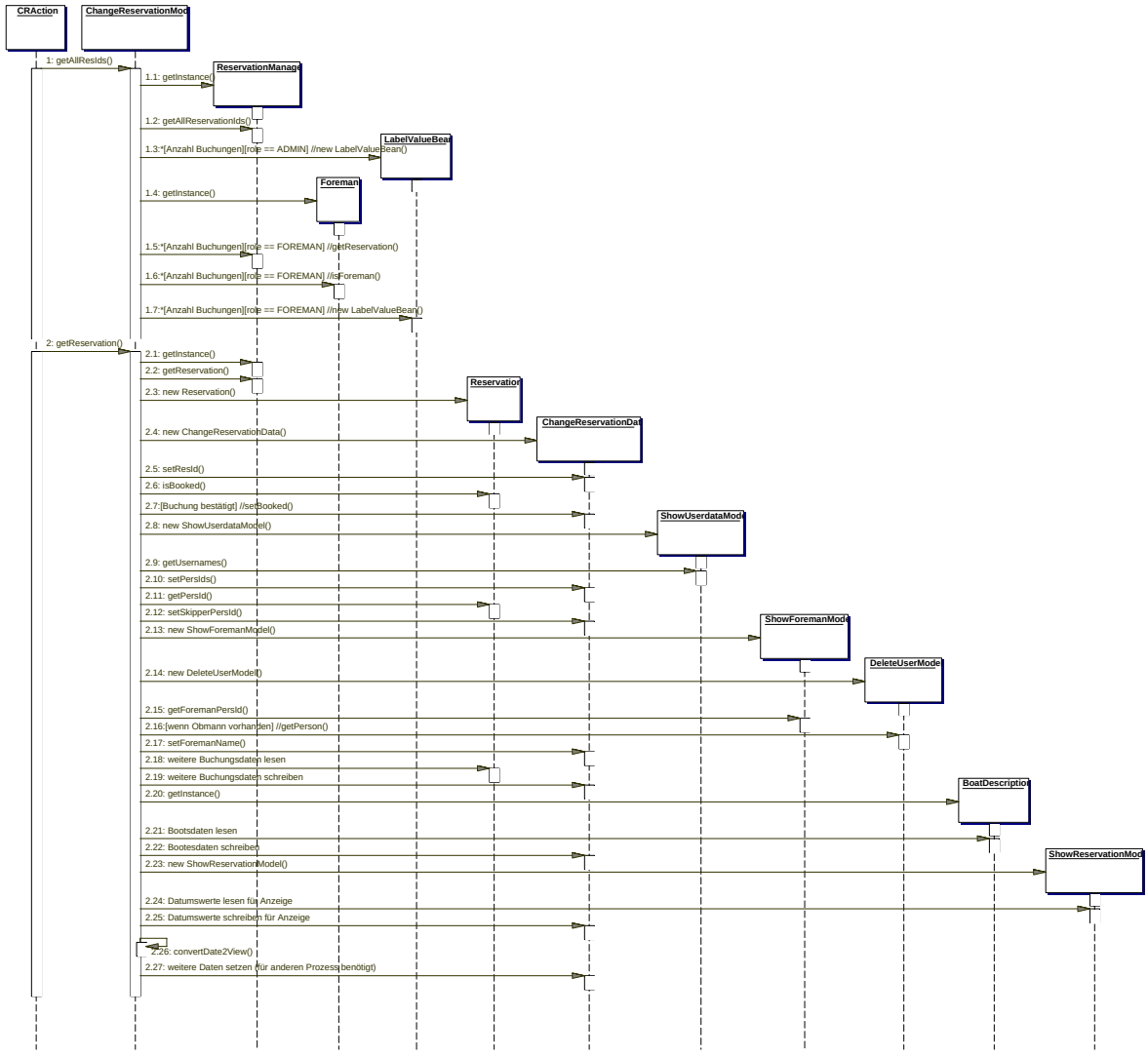


Abb. 50: Sequenzdiagramm Buchungsänderung durch Obmann (Model-Ansicht 1)

Der Begriff Model-Ansicht in der Beschriftung zu Abbildung 50 bezieht sich auf alles, was nach der Struts-Action kommt, also das Struts-Model und alle bekannten DiLog-Schichten (Kapitel 6.4.1). Abb. 50 zeigt nun, was genau passiert, wenn Aktion 1 in Abb. 49 ausgeführt wird. Aktion 2: getReservation() beschreibt die Vorgänge, die nötig sind um eine Buchung mit ihren Daten anzuzeigen. Dies korrespondiert mit Aktion 2 in Abb. 49 (Buchung öffnen). Aktion 3 im selben Diagramm (Buchung speichern) wird wie folgt ausgeführt:

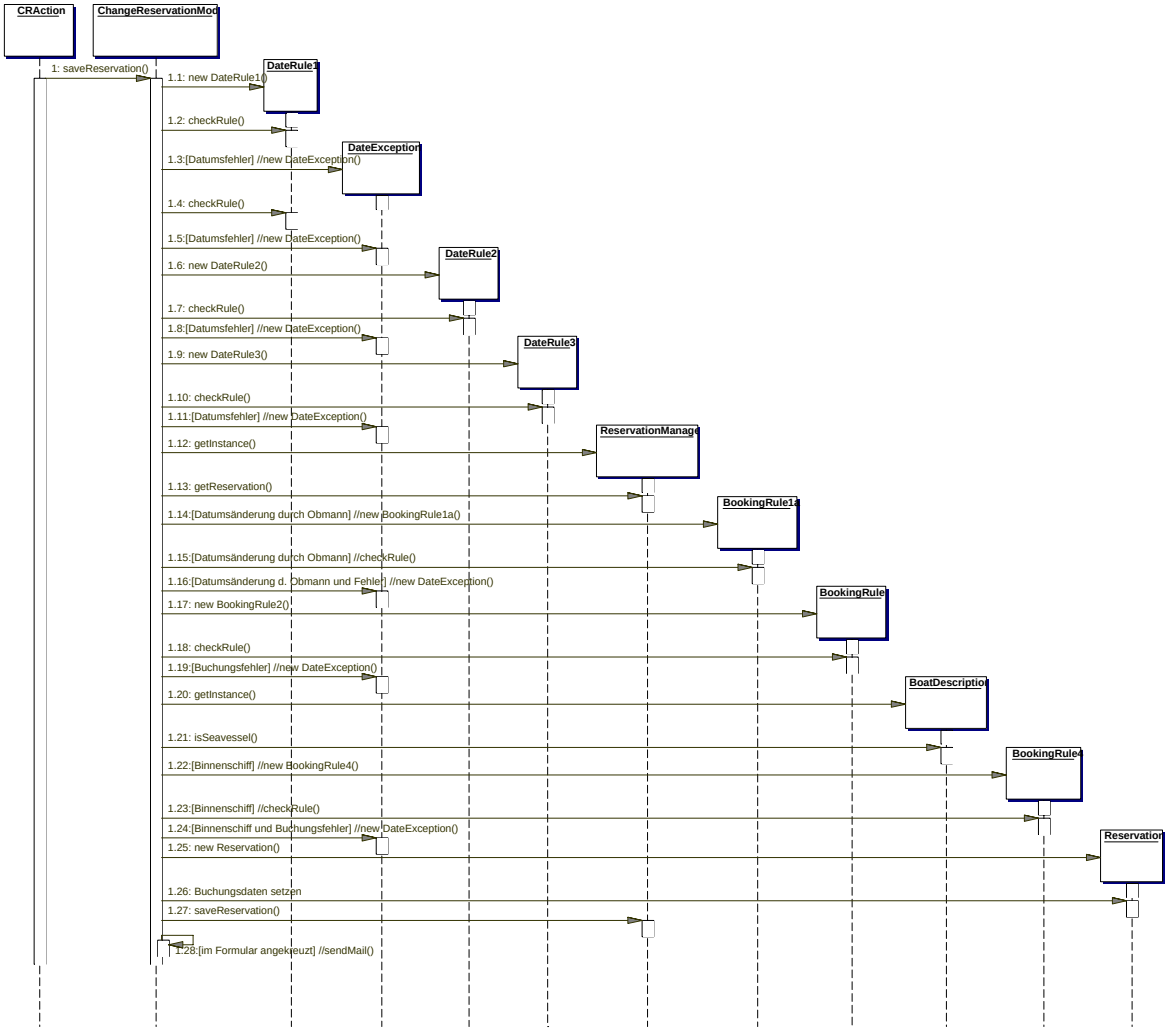


Abb. 51: Sequenzdiagramm Buchungsänderung durch Obmann (Model-Ansicht 2)

Vor Speicherung der Buchung muss geprüft werden, ob der zum Skipper ausgewählte Benutzer die entsprechende Segelberechtigung besitzt, ob das Boot für den gewählten Zeitraum verfügbar ist und ob die Buchung gültig ist, sprich korrekte Datumsangaben verwendet und Buchungsfristen einhält. Dies wird mit Ausführung sog. Rules erreicht. Jede Rule-Klasse überprüft nur eine bestimmte Gegebenheit. Dadurch können zusätzliche Regeln relativ einfach (im Prinzip modular) hinzugefügt werden.
War alles korrekt, erfolgt die Speicherung der Buchung: Dazu wird `saveReservation()` im `ReservationManager` aufgerufen, was veranlasst, dass alle Daten in die Datenbank geschrieben werden. Aktion 1.28: sendMail() wird nur ausgeführt, wenn der Obmann dies wünscht (zu sehen in Abb. 52 als Aktion 2: sendMail()).
Um ein weiteres Schaubild einzusparen, dient Abbildung 52 auch gleichzeitig zur Demonstration des Mail-Packages (Kapitel 6.7).

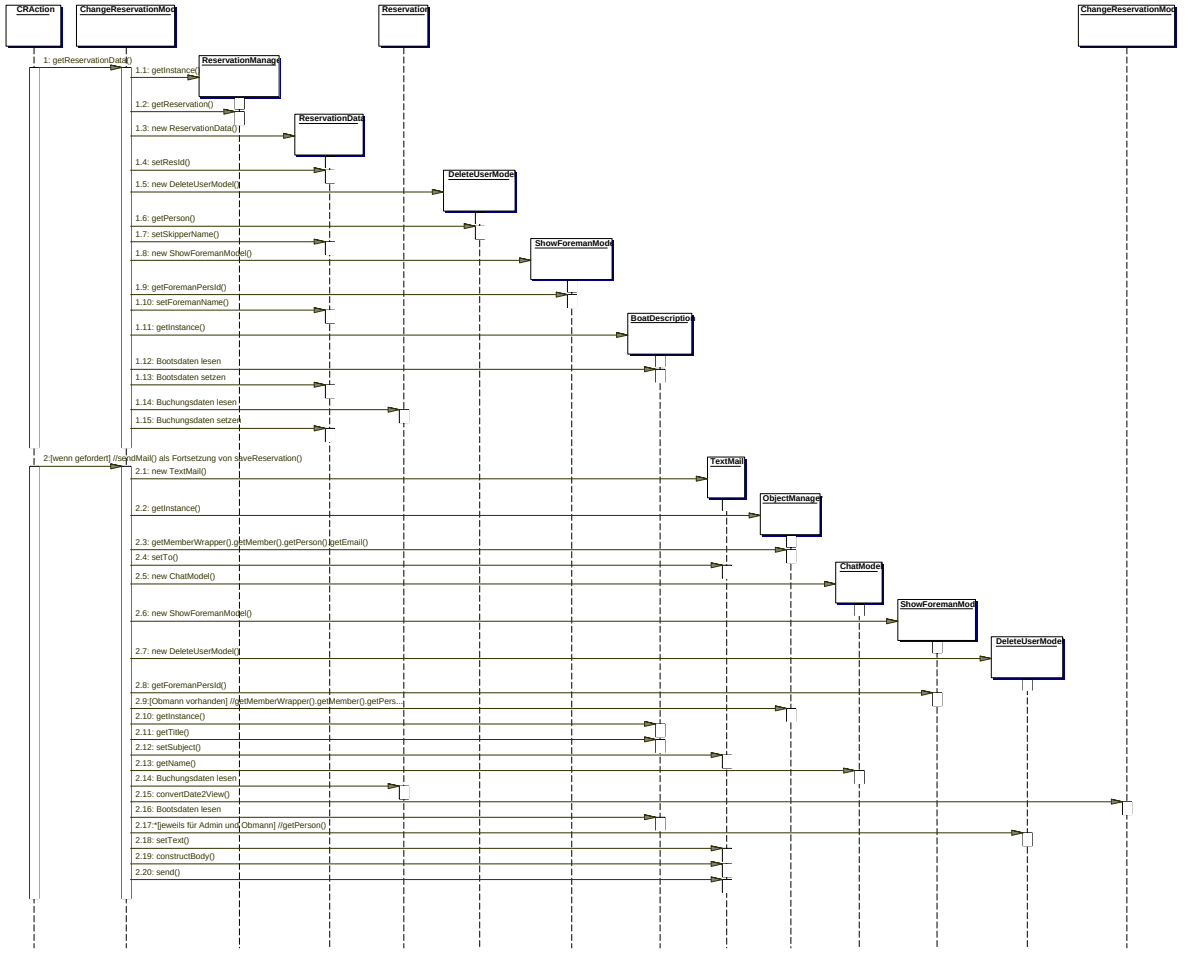


Abb. 52: Sequenzdiagramm Buchungsänderung durch Obmann (Model-Ansicht 3)

Aktion 1: getReservationData() im obigen Diagramm besorgt sich die geänderten Buchungsdaten vom `ReservationManager` um sie der Session als Bean hinzuzufügen und in der View auf einer Bestätigungsseite anzuzeigen.
Sicher wurde anhand dieses Geschäftsvorfalls (von dessen Komplexität viele weitere existieren) deutlich, dass es völlig unmöglich ist, alle Prozesse darzustellen, geschweige denn den Source Code detailliert zu erklären. Deshalb die Beschränkung auf fünf Beispiele, jeweils nur mit Sequenz- und evtl. Klassendiagramm dargestellt.

6.6.5 Administrator-Package

Als Beispiel für eine Administrator-Action wurde der Prozess „Initial-Logindaten für einen Benutzer erstellen" ausgewählt. Das Diagramm ist zweiteilig, Teil 1 beschreibt die Sicht auf die Struts-Action mit ihren Verbindungen, Teil 2 ist der Erzeugung der Initial-Logindaten und deren Einbettung in ein PDF verpflichtet (Aktion 3.5 in Abb. 53).
Was beschreibt das erste Diagramm? Zuerst werden wie üblich die Daten aus dem Backend zur Anzeige gebracht, wobei die Daten hier aus einer Auflistung aller im System vorhandenen Benutzer bestehen. Die Methode `execute()` wird ein zweites Mal aufgerufen, wenn der Admin einen Benutzer ausgewählt hat. Dabei werden Benutzerdaten gelesen und an die View geliefert. Im dritten Schritt wird sicherheitshalber die Autorisierung des

Administrators überprüft. Dann werden die weiteren Schritte zur Generierung der Logindaten und des PDFs ausgeführt.

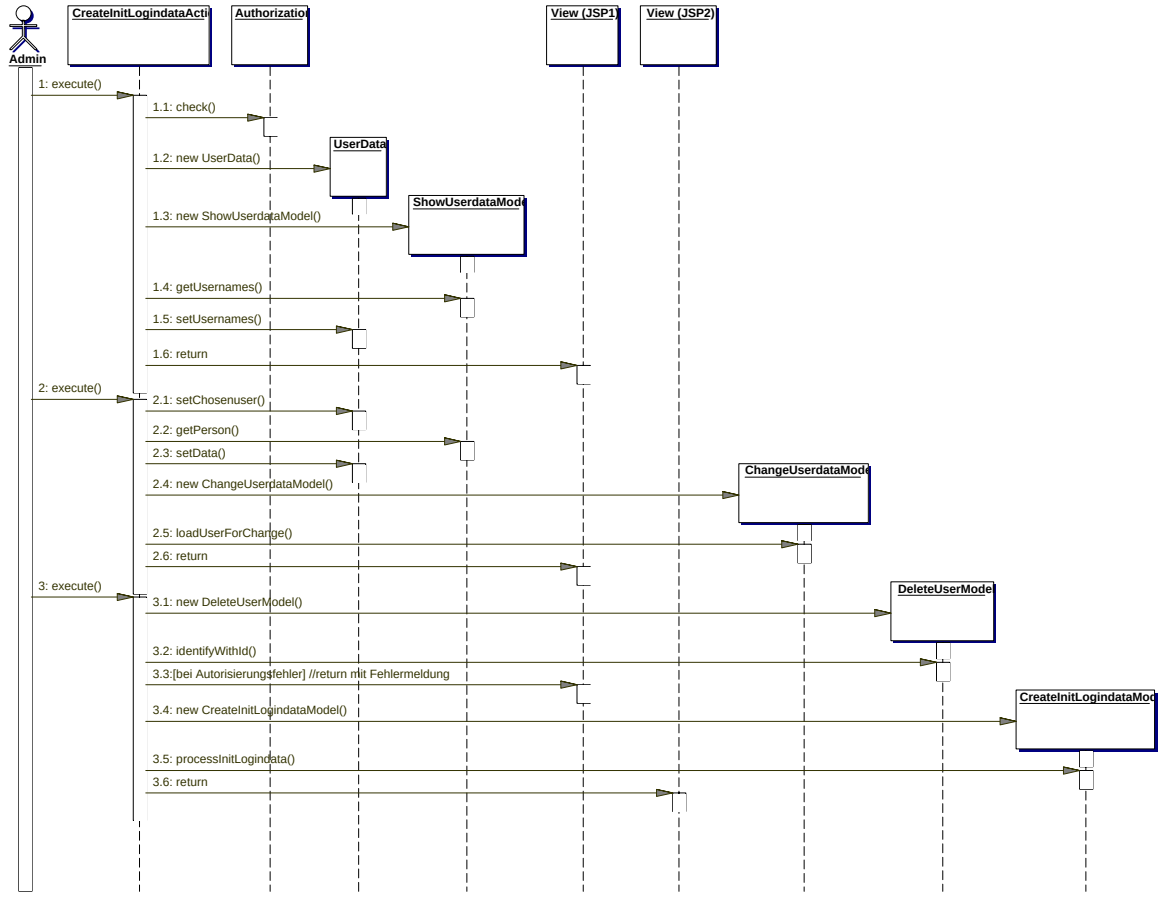


Abb. 53: Sequenzdiagramm Initial-Logindaten erstellen (Struts-Ansicht)

Im nachfolgenden Sequenzdiagramm wird die Methode `processInitLogindata()` der Klasse `CreateInitLogindataModel` detailliert beleuchtet. Innerhalb dieser Methode werden zuerst die Benutzerdaten in Form des MemberWrappers besorgt. Zur Erzeugung eines Passworts stellt die Klasse `magtime.helper.Admin` Funktionalität bereit. Dabei greift sie auf den `PasswordThread` zurück, welcher aus der aktuellen Systemzeit ein Passwort generiert. Die Erzeugung der Passwörter wurde in einen extra Thread ausgelagert (wird von Klasse `Admin` gerufen), weil die benutzte Klasse `Random` nach der Erzeugung eines Zufallswertes mehrere Millisekunden pausieren muss um nicht zu einem zweiten identischen Wert zu kommen. Dies ist wichtig bei der Erzeugung der Passwörter für alle Benutzer in einem Durchgang.

Nach der Erzeugung des Passworts wird es in die Datenbank gesichert (Aktion 2: init()). Anschließend folgt die Generierung des Anschreibens zur Mitteilung der neuen Initial-Logindaten in Form eines PDFs (siehe Abbildung 27). Der Vorgang mit seinen verschiedenen Klassen und Objekten wird in Kapitel 6.8 (PDF-Package) etwas näher betrachtet. Mit Aktion 12 wird der zuvor für die Ermittlung der Adressdaten für das PDF geladene `MemberWrapper` wieder entfernt, falls der entsprechende Benutzer nicht online ist.

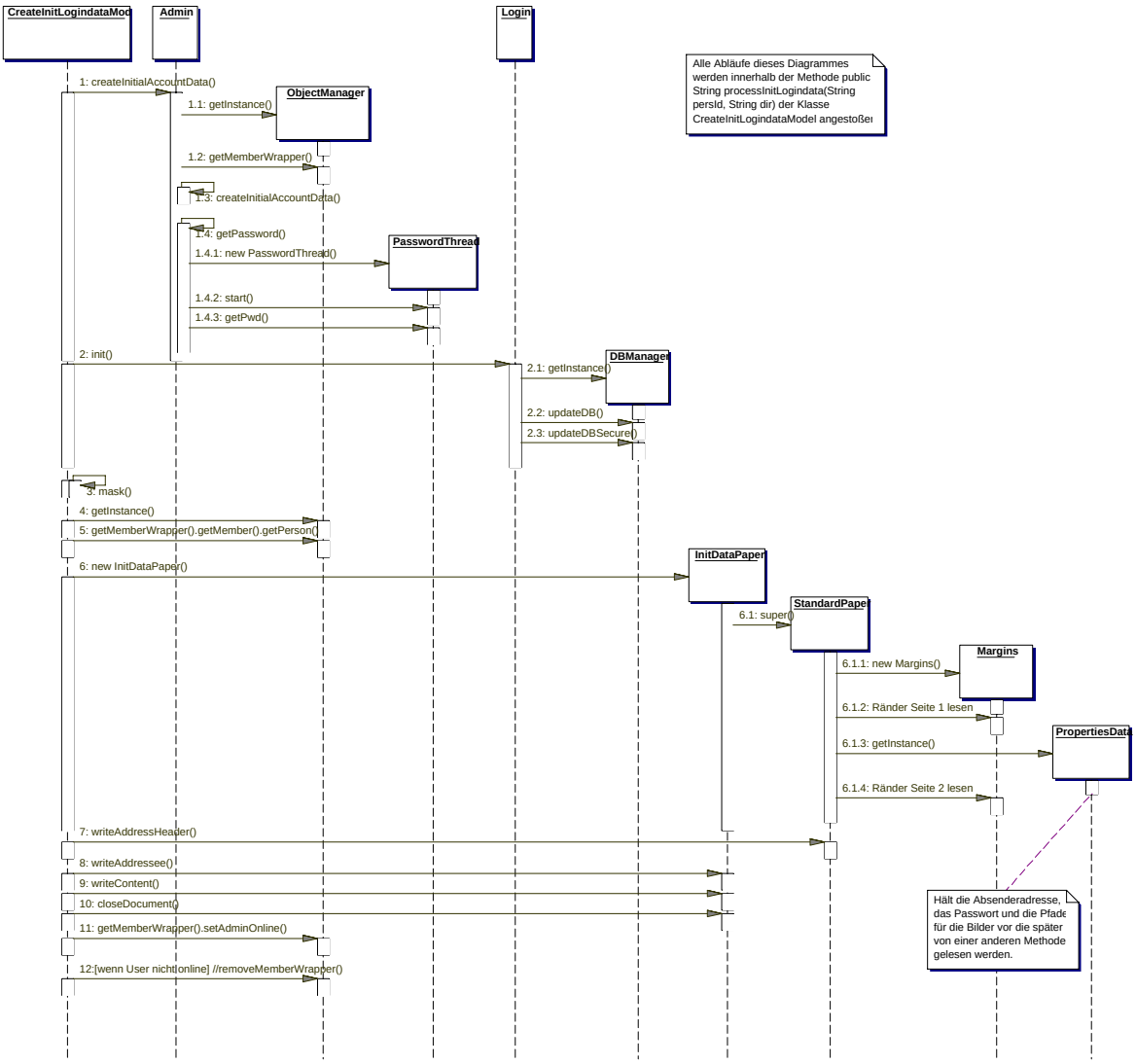


Abb. 54: Sequenzdiagramm Initial-Logindaten erstellen (Model-Ansicht)

6.6.6 Bean-Package

Zu den Data-Beans ist nicht weiter viel zu sagen: Sie bestehen hauptsächlich aus Gettern und Settern, werden zum Transport von Anzeigedaten vom Model zur View verwendet und erweitern sich zum Teil untereinander. Ein Klassendiagramm befindet sich auf der mitgelieferten CD-ROM.

6.6.7 Taglibs

Die Funktionsweise von Taglibs wurde im Kapitel 4.4.6 (und im Kap. 4.4.7) bereits erklärt. Was Struts-Tags angeht, so beantwortet die offizielle Dokumentation[54] weiterführende Fragen. Hier wird nur kurz auf die selbsterstellte Taglib Bezug genommen. Es folgt das Klassendiagramm für das Package `magtime.struts.taglibs.memberwrapper`.

54 http://struts.apache.org/userGuide/index.html, Links unter Developer Guides

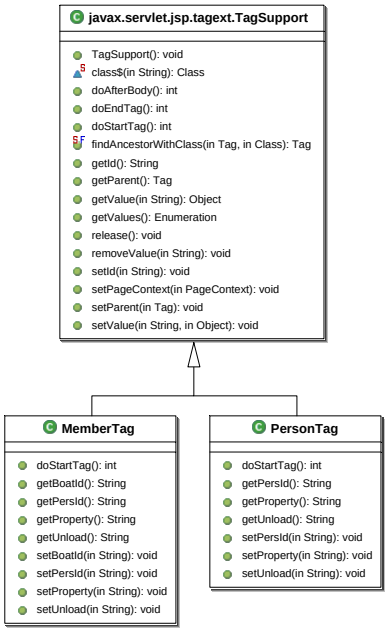


Abb. 55: Klassendiagramm Package magtime.struts.taglibs.memberwrapper

Alle eigenen Tags müssen die Klasse `TagSupport` aus dem Package `javax.servlet.jsp.tagext` erweitern. Wie Taglibs registriert und benutzt werden, ist bekannt.

6.6.8 Helper-Package und Sonstiges

Im Struts-Helper-Package befinden sich wie im Package `magtime.helper` allerlei nützliche Hilfsklassen. `SessionTimeout` zur Erledigung von Aufräumarbeiten und `BaseActionForm` als Oberklasse aller Form-Klassen wurden bereits öfters erwähnt. Die Klasse `Authrization` prüft, ob ein Benutzer eine bestimmte Struts-Action ausführen darf. Sie enthält nur die Methode `check()`, welche eine Weiterleitung in Form der Klasse `ActionForward` zurückgibt. Wenn die übergebene User-Session keine `persId` enthält, bedeutet dies, dass die Seite aufgerufen wurde, ohne dass sich der Benutzer vorher über die Login-Seite eingeloggt hat. Ergo muss der Zugang verweigert werden, was durch die Rückgabe der Weiterleitung auf die globale Weiterleitung mit dem Namen „notauthorized“ geschieht. Die zweite Prüfung ermittelt, ob der Status des Users ausreicht, um die Seite aufrufen zu dürfen. Das Interface `SystemRoles` hält die Zuweisung von Status auf eine Ganzzahl. Je höher die Zahl, desto größer die Rechte. Ist also Status Benutzer gefordert (Level = USER = 5) und meldet sich ein Gast (GUEST = 1) an, so wird der Zugang verweigert, da `level` 5 größer ist als Rolle 1, bzw. `role` kleiner als `level` ist. Im umgekehrten Fall wird null zurückgegeben, was bedeutet, dass keine Weiterleitung an dieser Stelle der Action stattfindet, der Code also normal weiter abgearbeitet wird (`ActionForward af = Authorization.check(request, mapping, USER); if (af != null) return af;`).

```
public class Authorization implements SystemRoles {

   public static synchronized ActionForward check(
      HttpServletRequest request,
      ActionMapping mapping,
      int level) {

      if (request.getSession().getAttribute("persId") == null) {
         request.getSession().setAttribute(
            "destination",
            mapping.getPath() + ".do");
         // Zugang verweigert
         return mapping.findForward("notauthorized");
      }

      if (((Integer) request.getSession().getAttribute("role")).intValue()
         < level) {
         // Zugang verweigert
         return mapping.findForward("notauthorized");
      }

      // Zugang erteilt
      return null;

   }

}
```

Listing 18: Klasse Authorization zur Zugangskontrolle einer Action

Klasse `NoFunctionForm` wird zwar in der Struts-Konfigurationsdatei als Form-Bean definiert, aber nicht wirklich verwendet. `NoAction` ist auch nicht zwingend notwendig: Der Ausdruck

```
<action path="/showchangedpersondata" type="magtime.struts.helper.NoAction">
   <forward name="show" path="dilog.user.showchangedpersondata" />
</action>
```

könnte gleichwertig auch einfach so geschrieben werden:

```
<action path="/showchangedpersondata " forward=" dilog.user.showchangedpersondata " />
```

6.7 Mail-Package

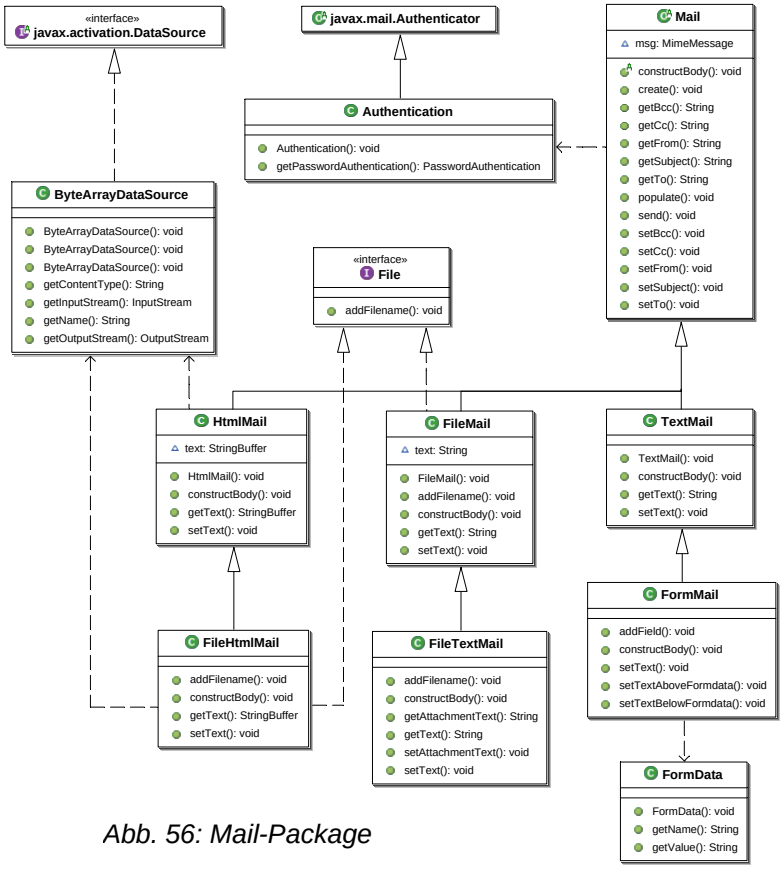


Abb. 56: Mail-Package

Das Mail-Package ist zuständig für die einfache Erstellung von unterschiedlichen Arten von Mails. Das Package bietet die Verwendung von `TextMail`, `HtmlMail`, `FileMail` und verschiedene Vermischungen (`FileHtmlMail`, `FormMail` und `FileTextMail`) an. Demzufolge ist die Erstellung von reinen Text-Mails, Mails, die HTML-Code unterstützen, und Mails mit Dateianhang recht einfach. Das Zauberwort bei den Mischformen lautet “multipart”. `FormMail` ist eine spezialisierte Text-Mail, denn sie benutzt Key-Value-Paare (gekapselt in `FormData`) um per E-Mail übermittelte Formulardaten grafisch etwas abgesetzt darzustellen.

Da alle Mail-Klassen die abstrakte Basisklasse `Mail` erweitern, können alle auch über die Klasse `Authentication` gesicherte Mailrelays von verantwortungsbewussten Providern benutzen. Das Mail-Package unterstützt SMTP[55]-Authentifizierung (Angabe von Kennung und Passwort vor dem Senden) also vollständig. Wie das Erstellen einer Mail genau vor sich geht, ist in Abb. 52 in Kapitel 6.6.4 in Form eines Sequenzdiagramms deutlich gemacht. Prinzipiell geht man wie folgt vor (Beispiel einer einfachen Text-Mail): `TextMail` instanziieren, Header-Daten der Mail setzen (`setTo()`, `setCc()`, `setBcc()`, `setFrom()`, `setSubject()`), den Inhaltstext angeben (`setText()`), anschließend Methode `constructBody()` aufrufen und Mail mit `send()` verschicken. Um den Rest kümmert sich das „Mini-Framework” und nicht der Anwendungsentwickler.

55 SMTP (Simple Mail Transfer Protocol) ist ein Protokoll zum Versenden von E-Mails über TCP/IP, das ohne gewisse Erweiterungen über keinerlei Sicherheitsmechanismen verfügt.

6.8 PDF-Package

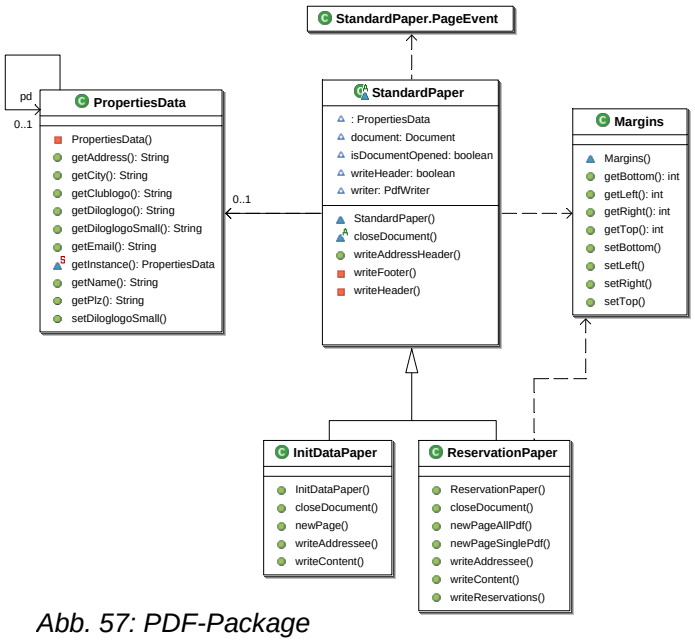


Abb. 57: PDF-Package

Auch hier gibt es ein zusätzliches Sequenzdiagramm zum links gezeigten Klassendiagramm, welches das schrittweise Vorgehen zur Konstruktion eines PDFs verdeutlicht: Die Aktionen 6 bis 10 in Abb. 54 erstellen das PDF mit den Initial-Logindaten.

Die Superklasse `StandardPaper` definiert eine abstrakte Methode und implementiert die Code-Abschnitte, die für alle PDFs in DiLog gleich sind. Dazu gehört die Definition des Headers und des Footers (Kopf- und Fußzeile), das Schreiben von Metadaten in den Header der PDF-Datei sowie das Verschlüsseln des Dokuments auf Wunsch. Das Passwort wird aus der Datei `clubaddress.properties` gelesen. Zudem werden im Konstruktor der Superklasse die Seitenrandeinstellungen für die erste Seite gesetzt, das Dokument zum Schreiben geöffnet und die Seitenränder für die weiteren Seiten definiert. Dies ist wichtig, weil ab der zweiten Seite kein Briefkopf mit Absender- und Empfängeradresse mehr geschrieben werden darf, sondern eine Kopfzeile mit dem ASV-Logo. Für die Aneinanderreihung von mehreren Einzel-PDFs zu einem Gesamt-PDF müssen die Einstellungen für die erste Seite aber trotzdem auch für nachfolgende Seiten geltend gemacht werden können. Mit der geerbten Methode `writeAdressHeader()` wird der Briefkopf geschrieben, `writeAddressee()` fügt die Empfängeradresse hinzu. In der Methode `writeContent()` wird aller Inhalt in Form von Textobjekten angegeben. Im Beispiel werden dazu die Klassen `Paragraph`, `Chunk`, `Anchor` und `FontFactory` der Bibliothek iText benutzt. Weitere Informationen zum Erstellen von PDFs mit iText enthält die Dokumentation und das hilfreiche mit der Bibliothek mitgelieferte Tutorial. Ein abschließendes `closeDocument()` schließt das PDF und schreibt den evtl. gepufferten Inhalt ins PDF.

Erwähnenswert ist noch die innere Klasse `PageEvent` in der Basisklasse aller DiLog-PDFs. Sie erweitert `PdfPageEventHelper` und ist in der Lage, Seitenereignisse bei der Generierung zu empfangen, was es dem Entwickler erlaubt, bei bestimmten Ereignissen spezielle dementsprechende Aktionen ausführen zu lassen. So werden beim Schreiben des PDF-Inhalts automatisch Seitenwechsel erzeugt, was mit der Methode `onStartPage()` bemerkt werden kann. Bei jeder neuen Seite werden so automatisch Kopf- und Fußzeile auf die Seite geschrieben, es sei denn, es handelt sich um die erste Seite des Dokuments.

6.9 Chat-Package

Dieses Package enthält Code für den Chat, unterteilt in Server-Klassen und Client-Klassen.

6.9.1 Chat-Server

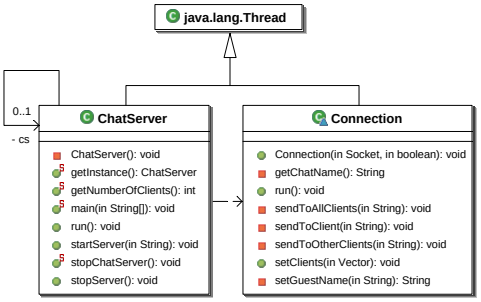


Abb. 58: Package Chat-Server

Der Chat-Server besteht nur aus zwei Klassen: Die von Thread abgeleitete Klasse `ChatServer` (verwaltet die ankommenden Socket-Verbindungen, startet und stoppt den Chat-Server) und `Connection`, welche die Socket-Verbindung darstellt und ankommende Nachrichten an andere registrierte Benutzer (= Connections) sendet. Kurz zu Funktion und Zusammenspiel der Klassen, bzw. Objekte:

Zum Starten des Chat-Servers besorgt sich die Struts-Model-Klasse mit `getInstance()`, wie beim Singleton-Pattern üblich, die Instanz der Klasse `ChatServer`. Auf diese Instanz wird die Methode `start()` aufgerufen, was den Thread mit Ausführung der `run()`-Methode startet. Er erzeugt ein `ServerSocket`, das in einer Endlosschleife (wird nur einmal pro neuem Client durchlaufen und blockiert dann) darauf wartet, dass eine Verbindungsanfrage von einem Client eingeht. Ist dies der Fall, wird eine neue `Connection` erzeugt und zum Pool der vorhandenen Clients, sprich Verbindungen, hinzugefügt. Im Konstruktor von `Connection` wird ein `BufferedReader` und ein `PrintWriter` erzeugt. Anschließend wird der Thread gestartet, denn auch `Connection` erweitert `Thread`. Zuerst wird der Benutzername eingelesen, den der Client an erster Stelle sendet. Über den `PrintWriter sockOut` wird die Nachricht an alle schon vorhandenen Clients verbreitet, dass sich ein neuer Client angemeldet hat. In folgendem Code-Ausschnitt ist der Programmfluss dann „gefangen" und hält die Kommunikation mit Hilfe des Readers und Writers aufrecht.

```
try {

   String zeile;
   while ((zeile = sockin.readLine()) != null) {
      sendToAllClients(name + ": " + zeile);
   }

} catch (Exception e) {

   logger.warn("Client nicht mehr erreichbar! (" + e.getMessage() + ")");

}
```

6.9.2 Chat-Client

Das `ClientApplet` wird durch den Browser gestartet und erstellt in der bekannten Methode `init()` das `Client`-Objekt. Dabei wird eine Referenz auf das Applet selbst übergegeben. Im Konstruktor werden grafische Komponenten erzeugt und sichtbar gemacht. Zudem erhält das Textfeld am unteren Ende der GUI einen `ActionListener`, der dafür sorgt, dass eingegebener Text beim Drücken der Enter-Taste über den `PrintWriter sockOut` an den Chat-Server gesendet wird. Zuvor muss dazu in der `connect()`-Methode aber eine Verbindung hergestellt werden. Es wird ein `Socket` aufgebaut

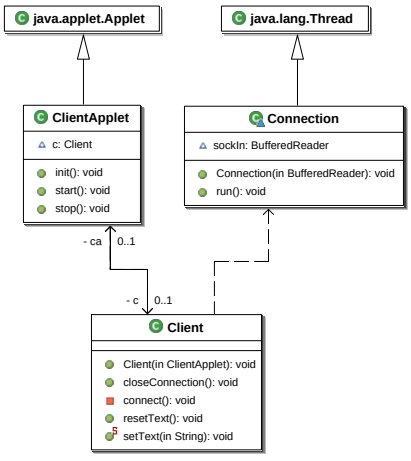


Abb. 59: Package Chat-Client

(entweder mit oder ohne Portangabe des Clients, siehe dazu auch 4.5.1 und 4.5.2), aus dem – wie auch beim `ServerSocket` – ein `InputStream` und ein `OutputStream` erzeugt wird, welche die Kommunikation mit anderen Clients über den Server übernehmen. Im `Connection`-Thread werden mit dem `BufferedReader` Nachrichten anderer Clients entgegengenommen und mit der statischen Methode `setText()` der Klasse `Client` auf das Ausgabefeld (eine `TextArea`) geschrieben.

6.10 Weitere Pakete (Webcam, Start, Testing)

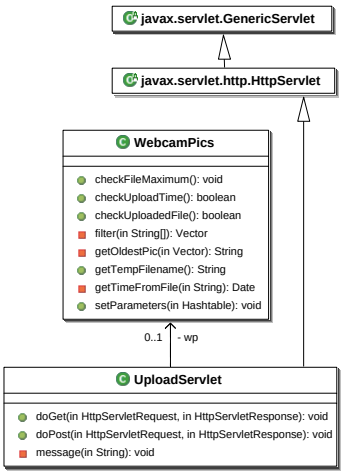


Abb. 60: Package webcam.server

Das Package `webcam.server` stellt das serverseitige Gegenstück zu dem in Kapitel 8 erläuterten Webcam-Client dar. Es nimmt die vom Webcam-Client hochgeladenen Bilder in Empfang und speichert sie so, dass der Prozess „Webcam-Bilder ansehen" die Bilder sortiert anzeigen kann. Das `UploadServlet` wird automatisch mit dem Start des Servlet-Containers gestartet und wartet ab diesem Zeitpunkt auf eingehende Requests. Die Methode `doPost()` benutzt den Commons-Fileupload. Der Commons-HttpClient sendet einen Multipart-Request, der Bilddaten, Kennung, Passwort und Datum enthält. Nun wird der Request geparst und auf Fehler überprüft. Sind beispielsweise keine zusätzlichen Daten außer dem Bild mitgesendet worden, so wird die Anfrage grundsätzlich verworfen.

Es folgt eine Iteration über alle FileItems in der Liste (ähnlich einer `Collection`-Klasse), wobei alle Schlüssel und Werte ausgelesen und in eine `Hashtable` geschrieben werden. Erst dann folgt die Authentifizierung der HTTP-Anfrage. Schlägt diese fehl, wird das zuvor temporär gespeicherte Bild augenblicklich gelöscht. Im Erfolgsfall wird überprüft, ob der Client das Zeitintervall eingehalten hat. Hat er zu früh gesendet, wird das Bild ebenfalls gelöscht. Sämtliche Überprüfungen nimmt ein Objekt der Klasse `WebcamPics` vor. Wenn alles korrekt verlief, dann wird die Datei umbenannt und im Ordner für die Webcam-Bilder abgelegt. Ein überzähliges Bild wird gelöscht, sodass die Struts-Action zum Anzeigen der Bilder immer genau 18 Stück vorfindet. Ein gültiger Bildname ist beispielsweise `webcampic_1104723901016.jpg`. Die Zahl entspricht den Millisekunden seit dem 01.01.1970. Bei der Anzeige wird diese „Metainformation" umgesetzt in die gewohnte Anzeige des Entstehungsdatums 03.01.05, 04:45:01 Uhr.

Auf das nicht zur Ausführung von DiLog benötigte Paket `magtime.testing` wurde in Kapitel 4.9 (Testen mit JUnit) schon recht ausführlich eingegangen. Deshalb gibt es an dieser Stelle dazu keine weiteren Informationen. Als sinnvoll und aufschlussreich erweist sich jedoch stets ein Blick in den Source Code.

Das Package `magtime.start` enthält Java-Applikationen ohne GUI wie das bereits erwähnte Werkzeug zur einmaligen Datenübernahme von Access nach MySQL (`StartDatatransformation`). Daneben existiert für jeden Teilabschnitt des regulären Datenimports oder –exports eine separate Startdatei. Diese waren während der

Entwicklungsphase von Bedeutung, wurden nun aber vom grafischen Import-Export-Tool auf Client-Seite und serverseitig von der Struts-Webapplikation abgelöst.
Als recht nützlich hat sich die Kommandozeilen-Applikation `ConfigCheck` erwiesen. Sie liest einfach alle XML-Konfigurationsdateien ein und parst sie mit dem Aufruf `cm.parseAll(true);`, wobei der Parameter angibt, dass eine Exception geworfen werden soll, wenn nicht alle bekannten Konfigurationsdateien angegeben wurden. Tritt beim Parsen ein Fehler auf, so wird dieser als Hinweis ausgegeben.

Abschließend fasst ein Komponentendiagramm die Zusammensetzung von DiLog und die räumliche Trennung seiner Komponenten noch einmal zusammen:

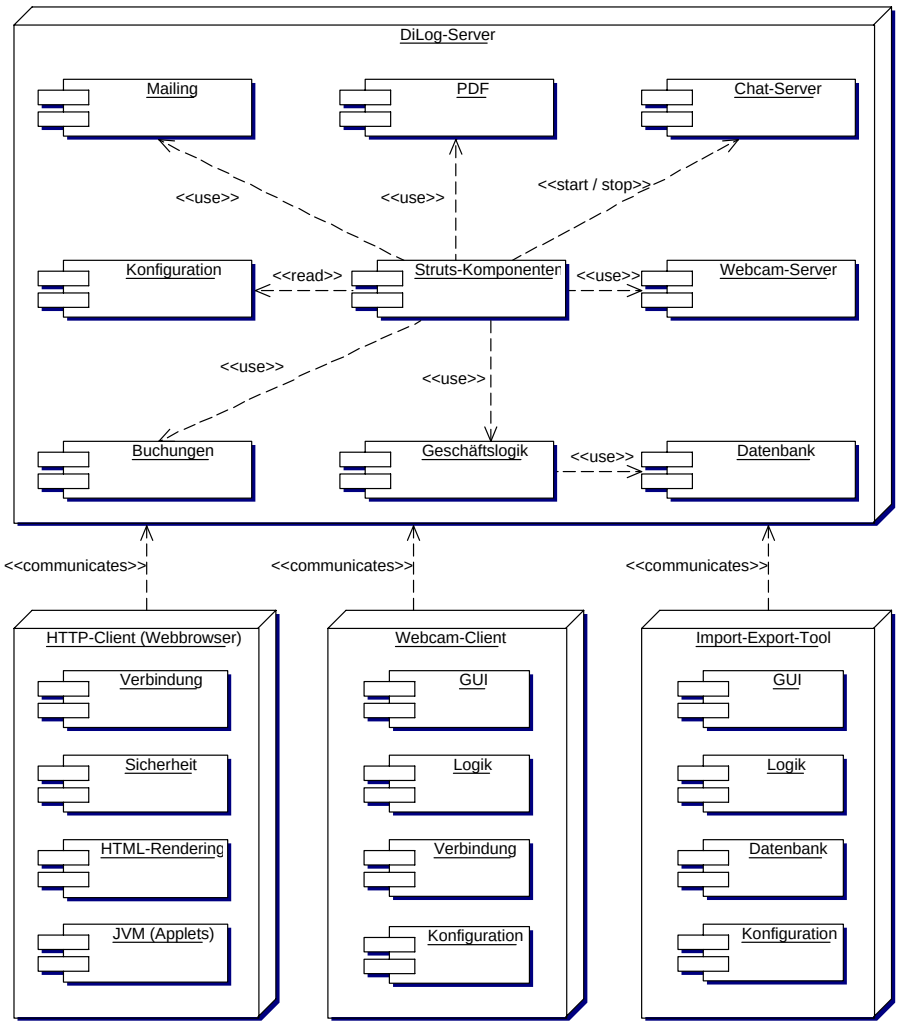


Abb. 61: Komponentendiagramm des Digitalen Logbuchs (DiLog)

7 Import-Export-Tool

Das IETool erfüllt zwei Aufgaben: Es importiert die gezippte XML-Datei, die aus MySQL exportiert wurde, und es exportiert die in Access geänderten Datensätze in eine gezippte XML-Datei.

7.1 Aufgaben

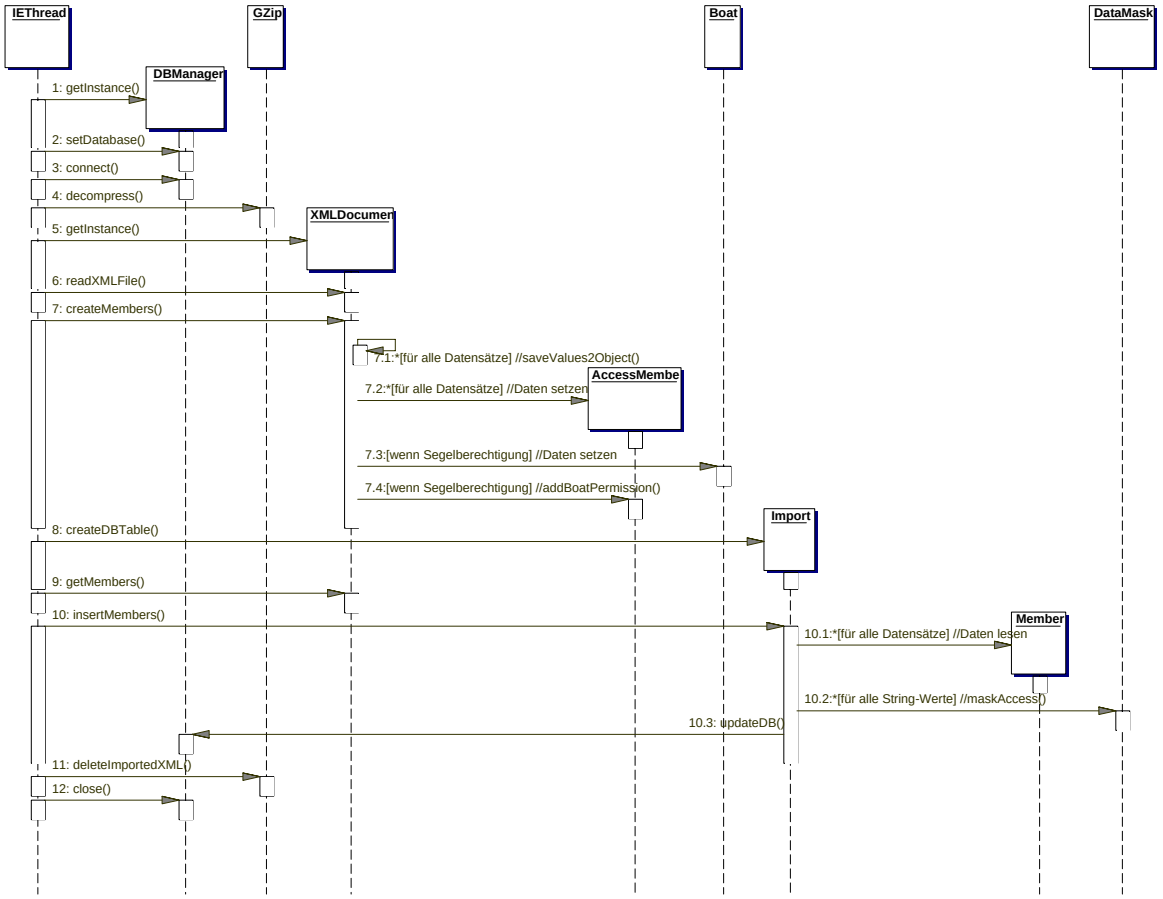


Abb. 62: Import von Mitgliederdaten nach Access

Grundsätzlich sieht der Ablauf ähnlich aus, wie in Abb. 33 und 34 schon gezeigt. Diese Ähnlichkeit ist nicht weiter verwunderlich, denn es werden dieselben oder ähnliche Klassen benutzt, und das Prinzip ist sowieso gleich. Aktion 4: decompress() entpackt die XML-Datei, anschließend wird sie eingelesen. Dazu wird pro Datensatz ein Objekt `AccessMember` erzeugt. Dann generiert das Programm die (unnormalisierte) Tabelle in der bei ODBC-Einstellungen angegebenen Access-Datenbank und schreibt alle Daten hinein. Je nach Konfigurationseinstellung wird die entpackte XML-Transportdatei hinterher gelöscht oder bleibt erhalten.

Das folgende Diagramm zeigt den weniger komplizierten Ablauf des Datenexports aus Access heraus in eine XML-Datei. Der grundsätzliche Vorgang hierbei dürfte inzwischen bekannt sein.
Erwähnenswert ist jedoch noch, dass vor dem Export die Datensätze je nach Änderung mit bestimmten Flags markiert werden müssen. Bei neu angelegten Datensätzen muss in die Spalte New der Wert 1, bzw. -1 für true eingetragen werden. Geänderte Datensätze werden in der Spalte Changed und zu löschende Datensätze in einer Spalte Del markiert. Die eigentliche Löschung findet erst beim Import der Daten in MySQL statt. Geschickterweise werden nur die Datensätze aus Access exportiert, die auch tatsächlich bearbeitet wurden. Das spart Zeit beim Export und Import sowie Bandbreite bei der Übertragung der Daten an den DiLog-Server.

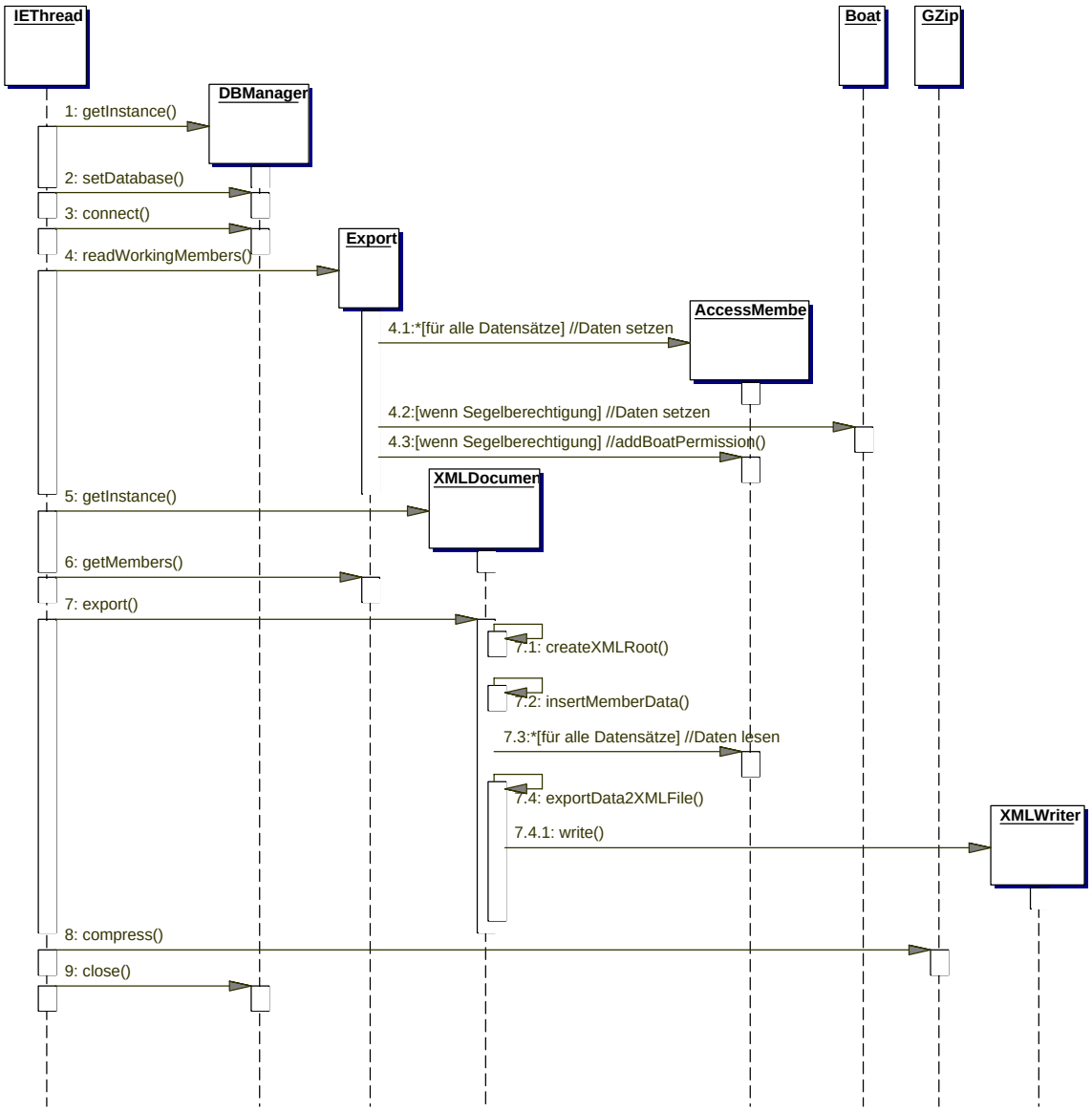


Abb. 63: Export von Mitgliederdaten aus Access

7.2 Struktur und Abläufe

Das nachfolgende Klassendiagramm beschreibt die Klassen des IETools und ihre Beziehungen untereinander. Der Aufbau ist relativ ähnlich wie beim im nächsten Kapitel beschriebenen Webcam-Client.
Dabei ist zu beachten, dass die Klassen für den eigentlichen Datenimport und –export nicht aufgeführt sind, da sie in den Packages `magtime.db.importaccessdb` und `magtime.db.exportaccessdb` beheimatet sind. Außerdem ähneln sie den Klassen für den Export aus und Import in MySQL.

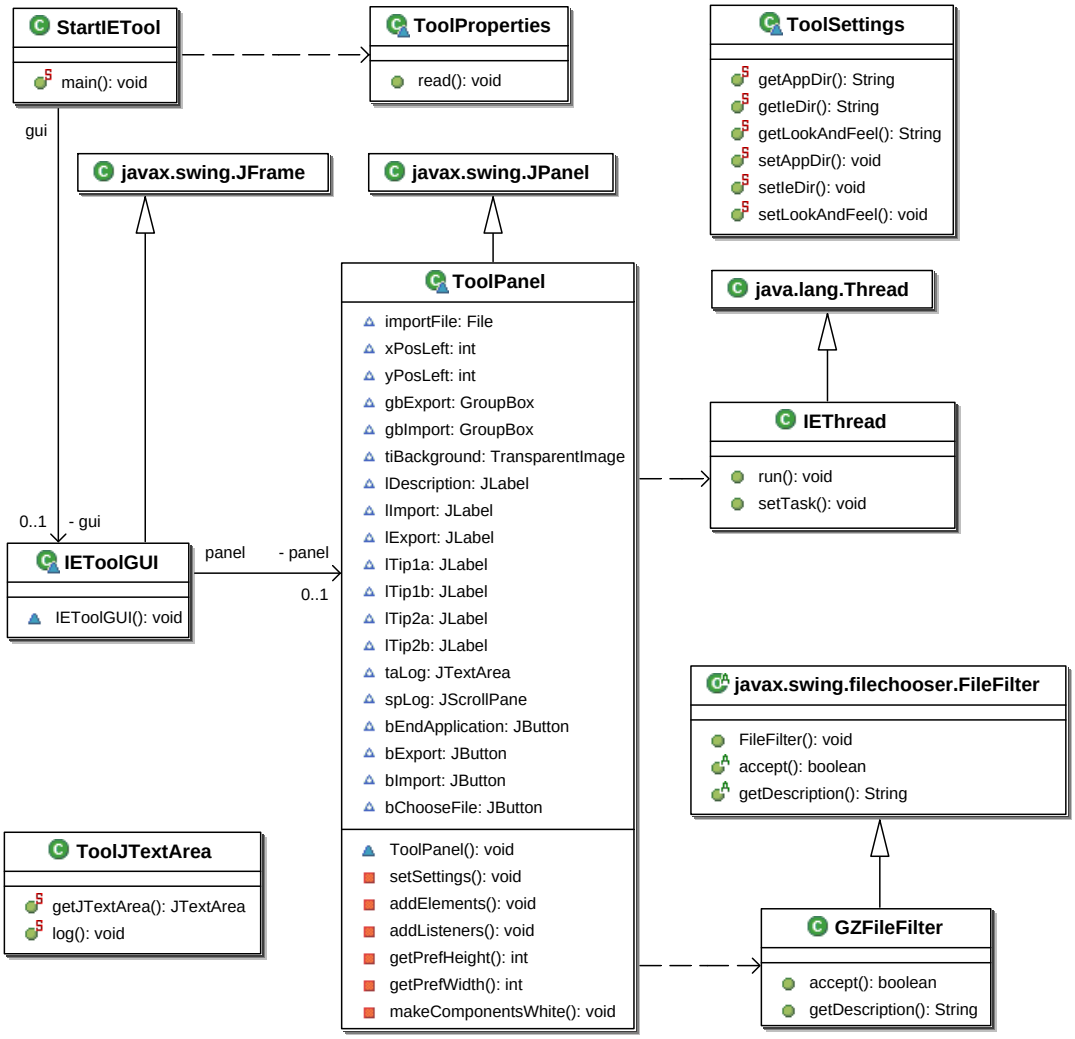


Abb. 64: Klassendiagramm Package magtime.ietool

Eine mitgelieferte und evtl. anzupassende Batch-Datei startet die Applikation. In `StartIETool` werden nach Initialisierung von Log4J mit dem Objekt `ToolProperties` die Konfigurationseinstellungen des IETools in die Beans `ToolSettings` und `SystemSettings` geschrieben. Es folgt das Einlesen und Parsen der üblichen XML-Konfigurationsdateien `boatconf.dilog.xml` und `dbconf.ietool.xml`. Anschließend wird das Hintergrundbild geladen, das Look & Feel ausgewählt und eine Instanz von `IEToolGUI` erzeugt und mit `show()` gezeigt.

Zwischen Erzeugung und Anzeige wird im Konstruktor von `IEToolGUI` (das von der Swing-Klasse `JFrame` erbt) eine Instanz von `ToolPanel` erzeugt. Diese `JPanel` erweiternde Klasse erzeugt die einzelnen Oberflächenelemente wie JButtons, GroupBoxes, JLabels, JTextAreas und JScrollPanes. Zudem werden den Schaltflächen Listener hinzugefügt, die auf die Betätigung derselben reagieren und weitere Prozesse anstoßen.
Die `ToolJTextArea` ist ein Wrapper für eine normale `JTextArea`, erweitert um die Funktionalität als Logging-Ausgabefeld zu fungieren. Nachrichten und Hinweise werden kumulierend ausgegeben, bis eine Zeichenanzahl von derzeit 5000 Zeichen überschritten ist. Wenn diese Anzahl erreicht wurde, dann wird der Inhalt des mehrzeiligen Textfeldes gelöscht.
Abbildung 65 fasst diese Vorgänge übersichtlich zusammen. Auf die Aktionen 11 bis 14 geht Kapitel 7.4 ein.

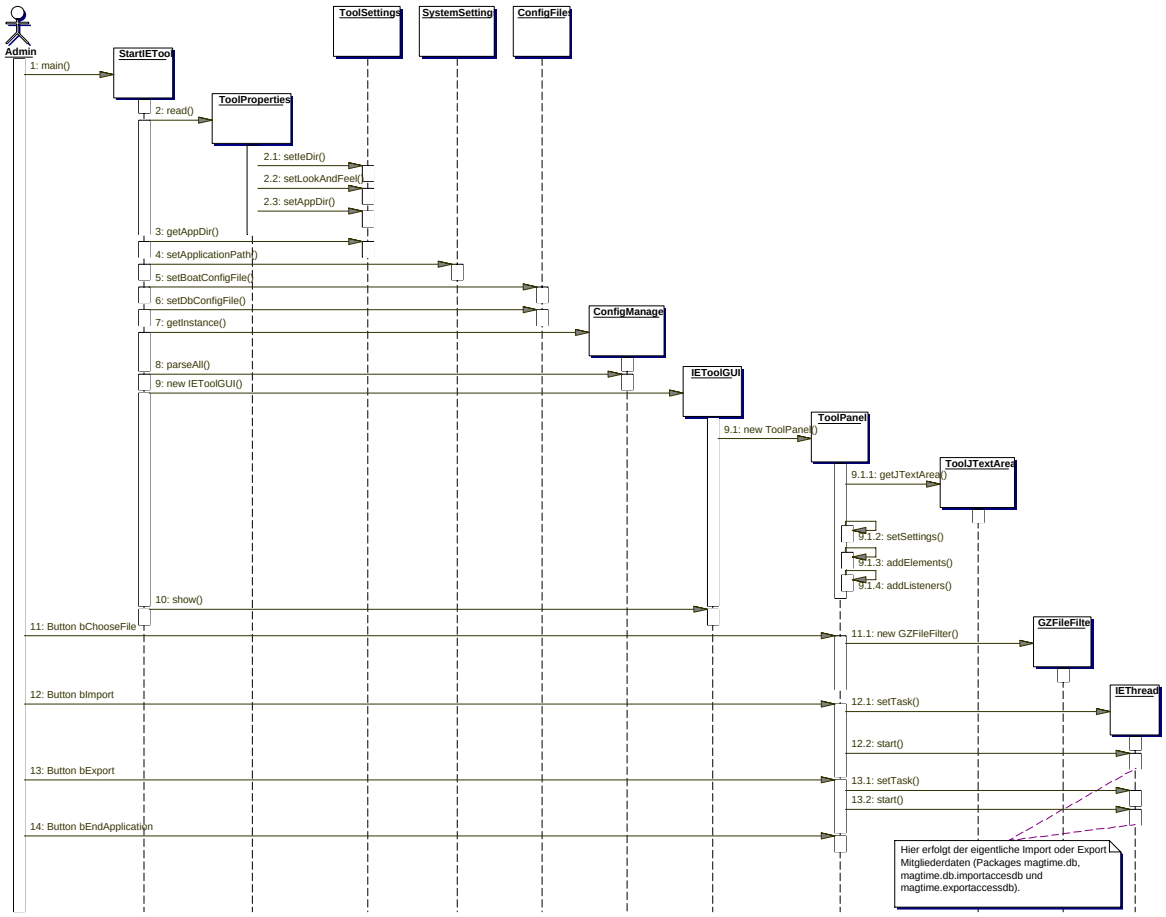


Abb. 65: Sequenzdiagramm eines Komplettdurchlaufs

7.3 Konfiguration

Die Konfiguration des IETools ist vergleichsweise harmlos. Aus der Datei `ietool.properties` werden die Angaben für Applikationspfad (vgl. Kap. 5.8.5), Import-Export-Verzeichnis und die Angabe für das Look & Feel (Windows | Metal) gelesen. Wenn das Verzeichnis für die Speicherung der temporären Dateien beim Import oder Export nicht angegeben wird, so benutzt das IETool dazu den Applikationspfad.

7.4 Benutzeroberfläche und Threads

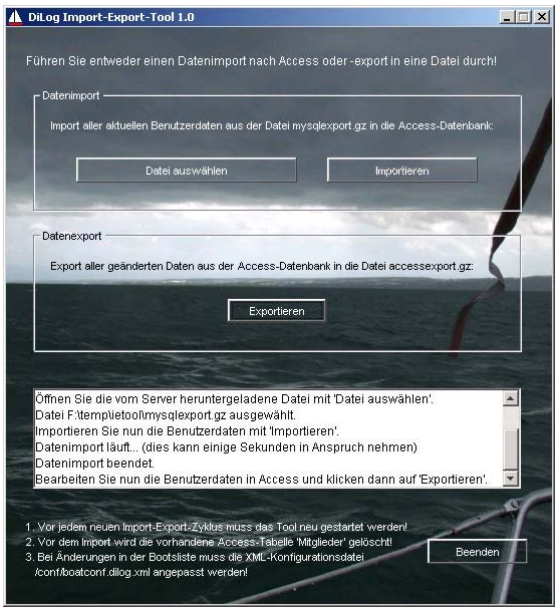


Abb. 66: Benutzeroberfläche des IETools

Wie zu sehen ist, besteht die Benutzeroberfläche aus mehreren Buttons und einer zentralen Textarea. In die Textarea werden Informationen darüber, was das Programm gerade tut, und Aufforderungen an den Benutzer geschrieben.

In zwei Groupboxes unterteilt befinden sich zwei Buttons, die den Import-Vorgang regeln und ein separater, der den Datenexport ermöglicht. Jeder Button verfügt über einen Listener. Wird die Schaltfläche „Beenden“ betätigt, so reagiert der Listener und führt nur die einfache Code-Zeile `System.exit(0);` aus. Der `ActionListener` für `bChooseFile` (Button „Datei auswählen“) ist dagegen etwas komplizierter gestrickt: Es wird ein `JFileChooser` erzeugt und konfiguriert (Titel, angezeigtes Verzeichnis, Button-Text, Datei-Auswahlverfahren), ein `GZFileFilter`-Objekt erstellt (mit dem Filter .gz/.GZ ausgestattet) und anschließend wird der Dateifilter dem `JFileChooser` übergeben. Somit zeigt das sich öffnende Dialogfenster zur Dateiauswahl nur Dateien an, welche die Endung .gz oder .GZ besitzen. Der vom Benutzer über diesen Dialog angegebene Dateipfad wird mit `importFile = fileChooser.getSelectedFile();` gespeichert und steht dem `IEThread` zur Verfügung.

Der `IEThread` wird erzeugt, wenn der Button „Importieren“ betätigt wurde. Geschieht das, bevor eine Datei ausgewählt wurde, erscheint eine entsprechende Fehlermeldung im Logging-Textfeld. Der extra Thread zum Datenimport (oder auch Datenexport) ist deshalb notwendig, weil ohne ihn der Kontrollfluss auf den Datenimport überginge und die Benutzeroberfläche so lange nicht reagieren würde. Das würde sich z.B. am gedrückten Button „Importieren“ bemerkbar machen: Die Schaltfläche bliebe gedrückt und würde erst wieder „herausspringen“, wenn der Datenexport beendet ist. Dies könnte beim Benutzer den Eindruck entstehen lassen, dass das Programm hänge oder gar abgestürzt sei. Ebenso könnte das Logging-Textfeld so lange keine Hinweise über den Status ausgeben.

Mit der Methode `setTask(IEThread.IMPORT, importFile)` wird dem `IEThread` über eine Konstante mitgeteilt, dass ein Datenimport ansteht. Gleichzeitig wird der Dateipfad der Importdatei übergeben. Anschließend wird der Thread gestartet und die MySQL-Daten werden importiert. Analog wird beim Datenexport verfahren…

8 Webcam-Client

Der Webcam-Client ist ein zusätzliches Java-Programm, das auf einem Rechner mit angeschlossener Webcam (in der Nähe des Segelreviers) installiert ist. Um die in einstellbaren Zeitabständen aufgenommenen Bilder an den DiLog-Server im Internet zu übertragen bedarf es eines Internet-Zugangs (Standleitung, DSL, ISDN oder analoger Zugang). Theoretisch sollte jede Webcam mit dieser Java-Applikation funktionieren, getestet wurde es jedoch nur mit einer Webcam (Logitech ClickSmart). Zur Zeit werden nur Bilder in einer Auflösung von 320 x 240 Pixel unterstützt.

8.1 Struktur und Abläufe

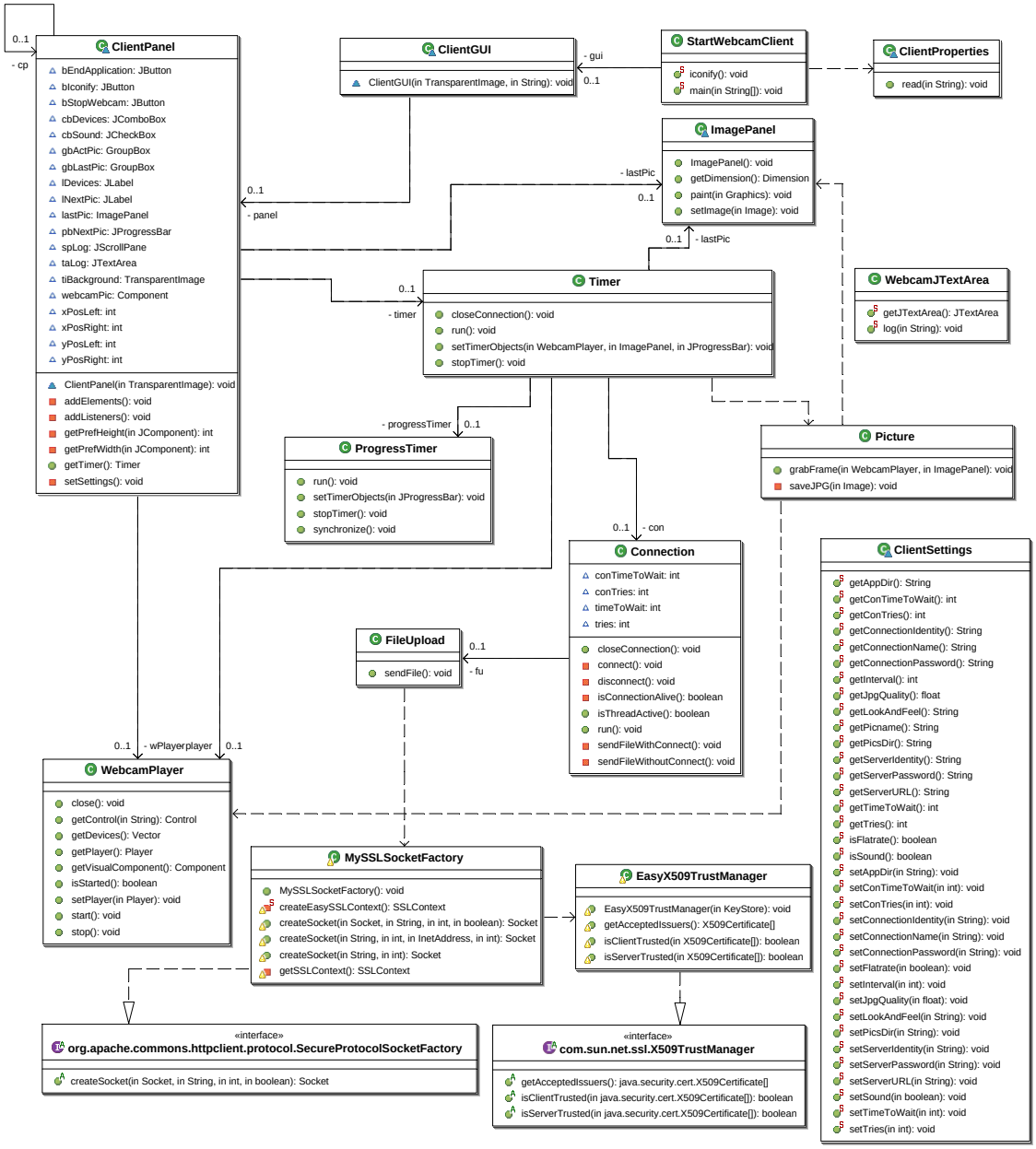


Abb. 67: Klassendiagramm Package magtime.webcam.client

Das Klassendiagramm zeigt alle für den Webcam-Client relevanten Klassen. Das Programm lässt sich bequem über die mitgelieferte Batch-Datei starten, wobei die Variable `param1` angepasst werden muss, denn sie gibt den Pfad des Webcam-Clients auf der Festplatte an. Das abgebildete Sequenzdiagramm enthält alle Abläufe vom Start der Applikation bis zu ihrer Beendigung. Nach Auswahl der Webcam startet automatisch ein Timer, der in einstellbaren Zeitintervallen Bilder der Webcam an den lokalen Rechner übermittelt und dann auf den DiLog-Server lädt. Dieser Kreislauf wiederholt sich so lange, bis der Timer gestoppt oder das Programm beendet wird.

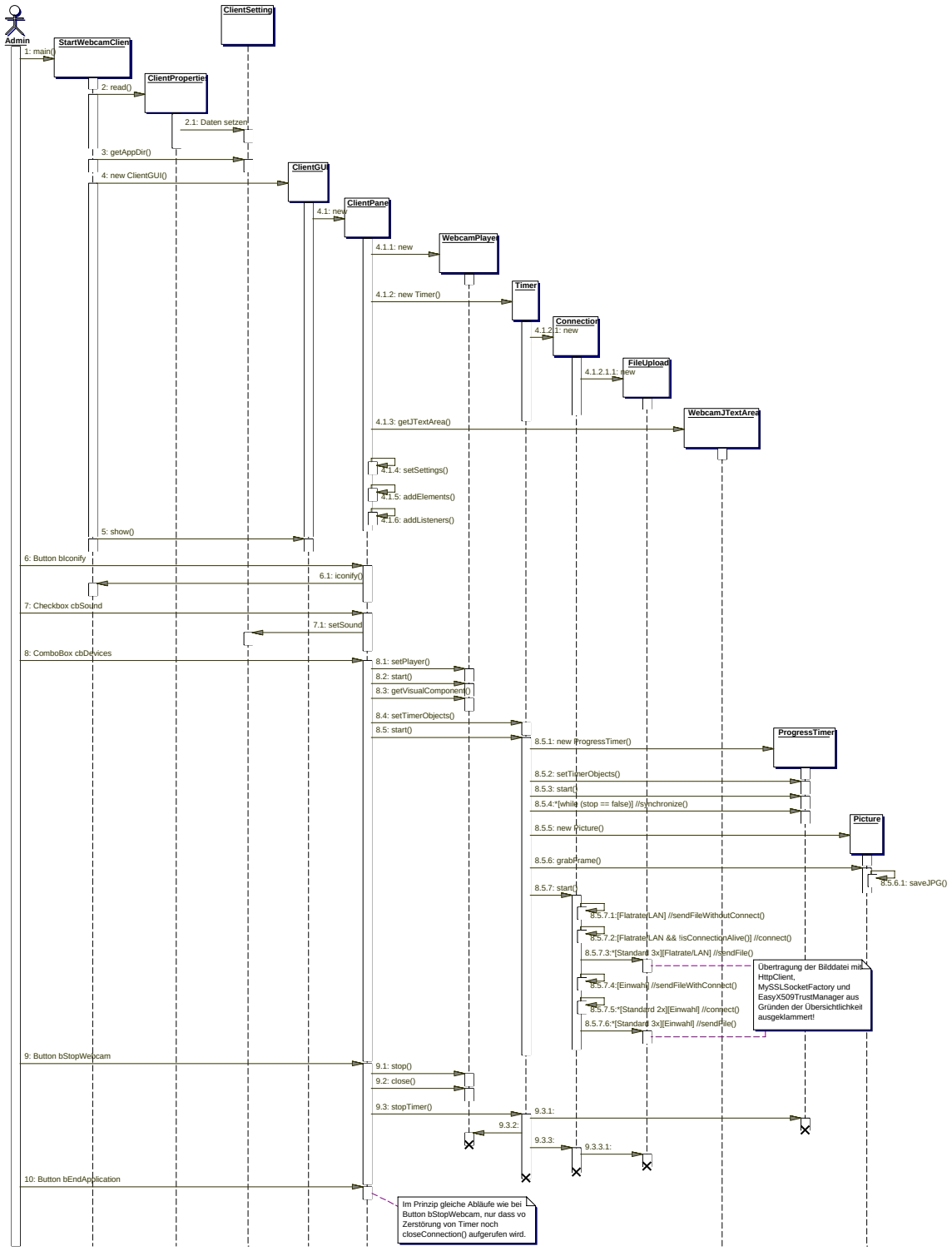


Abb. 68: Webcam-Client: Sequenzdiagramm eines Komplettdurchlaufs

`StartWebcamClient` enthält die `main()`-Methode, in der das Logging gestartet (Kapitel 4.8), die Konfiguration gelesen (Kapitel 8.2) und die grafische Oberfläche geladen wird (Kapitel 8.3). Sobald dies nach Aktion 5: show() im Sequenzdiagramm erledigt ist, wartet das Programm auf Benutzeraktionen (Events). Mögliche Ereignisse sind Auswahl, Start und Stopp der Webcam, Betätigen der Sound-Option und Verkleinerung oder Beendigung der Applikation. Empfängt eine Komponente ein solches Ereignis über ihren Listener, so wird die zugehörige Ausführung gestartet. Aufwändige Arbeiten wie z.B. der Verbindungsaufbau bei einer Einwahlverbindung und die Bildübertragung werden in einem extra Thread ausgeführt. Wäre dies nicht so, so würde die GUI einfrieren, bis die Aufgabe abgearbeitet ist.

8.2 Konfiguration

Die Log4J-Einstellungen werden wie gewohnt in der Datei `log4jconf.webcamclient.xml` getätigt. Für die reinen Applikationseinstellungen existiert die Datei `client.properties`, wofür die Datei `hinweise.txt` zusätzliche Tipps und Erläuterungen bereithält.
Über die Properties-Datei lassen sich folgende Dinge einstellen:

- Der Applikationspfad ermöglicht Webcam-Client benötigte Ressourcen zu finden.
- Look & Feel erlaubt das Umschalten zwischen den beiden mit Java mitgelieferten Stilen Windows und Metal.
- Qualität der JPEG-Bilder (höchste Qualität = 1.0 empfohlen).
- Pfad zum Zwischenspeichern der Bilder auf der lokalen Festplatte.
- Soundeinstellung und Bildübertragungsintervall. Hierbei ist zu beachten, dass der Wert (in Sekunden) größer sein muss als der beim DiLog-Server eingestellte Wert und drei Minuten (180) sinnvollerweise nicht unterschreiten sollte.
- Die Server-URL gibt das Übertragungsziel an. An diese Adresse (auf der ein Servlet wartet) werden Bilddaten, Kennung und Passwort übermittelt.
- Verbindungsdaten, die von der mitgelieferten EXE-Datei `rasdial95` benötigt werden. Dieses Programm kann auf allen Windows-Systemen im Hintergrund eine Verbindung über eine zuvor unter Netzwerkverbindungen angelegte DFÜ-Verbindung herstellen. Bei Linux-Systemen und anderen muss eine permanente (LAN-)Verbindung (Standleitung) bestehen.[56]
    - Flatrate gibt an, ob sich der Client mit `rasdial95.exe` einwählen soll oder nicht.
    - connectionName bezeichnet die zuvor angelegte DFÜ-Verbindung.
    - Kennung und Name müssen je nach Provider hinterlegt werden.
    - conTries gibt die Anzahl der Einwahlversuche an (z.B. bei besetzter Leitung oder Provider-Ausfall). conTimeToWait legt fest, wie lange bis zum nächsten Einwahlversuch gewartet werden soll.
    - tries gibt die Anzahl der Übertragungsversuche bei aufgebauter Verbindung an (wenn z.B. der DiLog-Server nicht reagiert). timeToWait legt fest, wie lange bis zum nächsten Übertragungsversuch gewartet werden soll.

[56] Der Webcam-Client wurde nur auf verschiedenen Windows-Systemen und -Plattformen getestet. Eventuelles Fehlverhalten bei der Kamerasteuerung und beim Senden der Bilder kann bei anderen Systemen nicht ausgeschlossen werden.

Alle Konfigurationsdaten in `client.properties` werden von der Klasse, bzw. einem Objekt der Klasse `ClientProperties` gelesen (2: read()) und in die Klasse `ClientSettings` geschrieben, die alle Eigenschaften mit statischen Methoden systemweit – also innerhalb der Webcam-Client-Anwendung – zur Verfügung stellt (2.1: Daten setzen). Damit ist die Konfigurierung des Webcam-Clients abgeschlossen.

8.3 Benutzeroberfläche und Threads

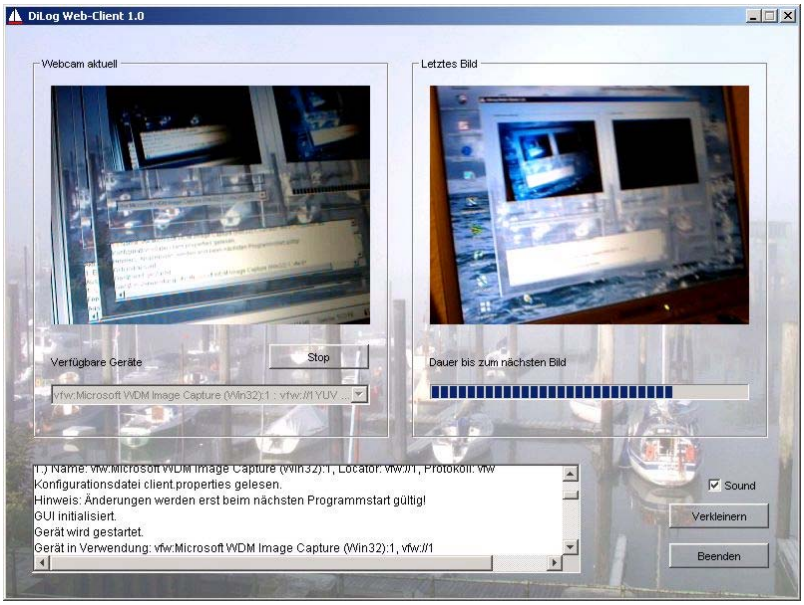


Abb. 69: Benutzeroberfläche des Webcam-Clients

Die als Abbildung 69 gezeigte Benutzeroberfläche besteht hauptsächlich aus mehreren Buttons, zwei Bildflächen, einer Combobox, einer Progressbar und einer Textarea. In die Textarea werden ständig Informationen darüber, was das Programm gerade tut, geschrieben. Das linke Bild ist eigentlich eher ein Videofenster, denn es wird wie in einem Film ständig das Bild aktualisiert. Rechts wird jeweils das zuletzt gespeicherte und übertragene Bild angezeigt.

Das Objekt `ClientGUI` setzt im Konstruktor einige Fensterwerte, fügt einen Window-Listener hinzu und hält eine Referenz auf das Objekt `ClientPanel`. Der Window-Listener gibt in einer anonymen inneren Klasse an, was passieren soll, wenn auf das x rechts oben auf der Programmoberfläche geklickt wird, nämlich das Schließen einer evtl. geöffneten Verbindung und das Beenden des Programms. `ClientPanel`, das die Swing-Klasse `JPanel` erweitert, bestimmt den überwiegenden Teil des Aussehens der Oberfläche, denn über den Konstruktor werden sämtliche Eigenschaften aller Standard-Swing-Oberflächenelemente gesetzt, zur Oberfläche hinzugefügt und, wo notwendig, mit Listenern versehen.

GUI-Element	***vom Listener ausgeführte Aufgaben***
bStopWebcam (1)	Stoppt und schließt den Player (für Webcam), modifiziert Schaltflächenzustände, stoppt Timer für Bildspeicherung und Verbindungsaufbau.
bEndApplication (1)	Stoppt und schließt den Player (für Webcam), stoppt Timer für Bildspeicherung und Verbindungsaufbau, schließt evtl. geöffnete Verbindung.
bIconify (1)	Verbirgt die Programmoberfläche und setzt sie in die Taskleiste.
cbSound (2)	Stellt Geräuschausgabe an oder aus.
cbDevices (3)	Initialisiert und startet Player mit der ausgewählten Webcam, fügt der Oberfläche die visuelle Player-Komponente hinzu, startet Timer, modifiziert Schaltflächenzustände.

Abb. 70: GUI-Elemente des Webcam-Clients und ihre Aufgaben

Die 1 in Spalte GUI-Element steht für Buttons, 2 für Checkboxes und 3 für Comboboxes. Die Klasse `WebcamJTextArea` kapselt die Swing-Klasse `JTextArea` und fügt ihr eine spezifische Logging-Methode hinzu, welche Ausgaben zu bereits bestehenden Ausgaben hinzufügt und das Ausgabefeld bei Überschreitung einer bestimmten Anzahl an Textzeichen (derzeit 10.000 Zeichen) löscht.

Die Klasse `ImagePanel` erweitert wie `ClientPanel` ebenfalls die Klasse `JPanel`. Von `ImagePanel` wird zwar gleich beim Start durch `ClientPanel` eine Instanz erzeugt, doch ist dessen Layout-Wert zu Beginn `null`, sodass nichts angezeigt wird. Erst wenn `Timer` das `Picture`-Objekt mit `grabFrame()` anweist, das aktuelle Bild aus dem Bilderstrom der Webcam anzuzeigen (`lastPic.setImage(img);`) und zu speichern (`saveJPG(img);`), wird das zu diesem Zeitpunkt im linken Rahmen der Oberfläche angezeigte Bild der Webcam auch rechts angezeigt.

Die Klasse `WebcamPlayer` ist gewissermaßen auch eine der grafischen Oberfläche zuzuordnende Klasse, denn sie stellt eine Wrapper-Klasse für den `Player` des Java Media Frameworks dar und kann somit eine grafische Komponente (`Component` aus AWT, Abstract Window Toolkit) zurückgeben. Dies passiert auch genau so, wenn der `Player` nach Betätigung der Combobox (und somit Aktivierung der Webcam) gestartet wird.

Wegen der geforderten parallelen Verarbeitung von Ereignissen und Aufgaben muss der Webcam-Client mehrere Threads gleichzeitig ausführen. Abbildung 71 benennt die Threads und ihre Aufgaben. Aus Klassendiagramm 67 und Sequenzdiagramm 68 sollte erkenntlich sein, wie die einzelnen Klassen, bzw. Objekte zusammenspielen.

Klasse	***Beschreibung des Threads***
Timer	Zentrale Ablaufsteuerung des Webcam-Clients. Hier werden die Bildaufnahme und dessen Speicherung, der Verbindungsaufbau und -abbau, der Start der Bildübertragung und die Aktualisierung des Fortschrittsbalkens angestoßen.
ProgressTimer	ProgressTimer ist der Thread, der dafür sorgt, dass der Fortschrittsbalken für die Anzeige der Dauer bis zur nächsten Bildspeicherung laufend aktualisiert wird. Dies geschieht alle 100 Millisekunden. ProgressTimer wird von Timer gestartet, sobald ein Bild aufgenommen wurde.
Connection	Regelt das Verbinden und Trennen, d.h., es sorgt dafür, dass der Webcam-Client online sein kann, wenn er ein geschossenes Bild an den Server übertragen will.

Abb. 71: Übersicht über Threads des Webcam-Clients

8.4 Kamerasteuerung mit JMF

Das Java Media Framework ermöglicht das Einlesen (Capture), Verarbeiten und Speichern von Audio- und Videosignalen von Mikrofonen und/oder Kameras. Zusätzlich sind Transformationen (z.B. WAV-Dateien in MP3-Dateien) und Streaming möglich. Es können verschiedene Ressourcen für die Ein- oder Ausgabe verwendet werden. Für die Ausgabe kommen beispielsweise Lautsprecher, Bildschirm oder Dateisystem in Frage. Die Steuerung dieser zeitbasierten Medien beruht auf sog. Managern. Durch Plugin-Technologie ist es

möglich, Ressourcen in Reihe zu schalten oder zur Datenmanipulation sog. Prozessoren zwischenzuschalten.

Beim Start des Webcam-Clients werden alle verfügbaren den gewünschten Kriterien (RGB) entsprechenden Geräte über einen Geräte-Manager eingelesen und ihre Gerätekennung ausgegeben:

```
Vector devices = CaptureDeviceManager.getDeviceList(new VideoFormat("RGB"));
if (devices == null || devices.size() < 1) {
   WebcamJTextArea.log("Keine Geräte gefunden!");
} else {

   WebcamJTextArea.log("Anzahl Geräte: " + devices.size());
   for (int i = 0; i < devices.size(); i++) {

      CaptureDeviceInfo cdi = (CaptureDeviceInfo) devices.get(i);
      MediaLocator ml = cdi.getLocator();
      WebcamJTextArea.log(1 + i + ".) Name: " + cdi.getName() + ", Locator: "
         + cdi.getLocator() + ", Protokoll: " + ml.getProtocol());

   }
}
```

Listing 19 zeigt den auf das Wesentliche reduzierten Listener (eine anonyme innere Klasse) für die Combobox, über die der Benutzer die gewünschte Webcam auswählen kann.

```
cbDevices.addActionListener(new ActionListener() {

   public void actionPerformed(ActionEvent ae) {

      WebcamJTextArea.log("Gerät wird gestartet.");
      CaptureDeviceInfo cdi = (CaptureDeviceInfo) cbDevices.getSelectedItem();
      MediaLocator ml = cdi.getLocator();

      try {

         wPlayer.setPlayer(Manager.createRealizedPlayer(ml));

      } catch (Exception e) {
         WebcamJTextArea.log("Unter Medialocator " + ml + " wurde kein Gerät gefunden!");
      }

      WebcamJTextArea.log("Gerät in Verwendung: " + cdi.getName() + ", "
         + cdi.getLocator());

      try {
         wPlayer.start();
      } catch (Exception e) {

         WebcamJTextArea.log("Keine Verbindung zum angegebenen Gerät möglich! Evtl. läuft
            Webcam-Client schon im Hintergrund?! Wenn ja, beenden Sie bitte den anderen
            Webcam-Client!");
         return;

      }

      webcamPic = wPlayer.getVisualComponent();
      cp.add(webcamPic);
      cbDevices.setEnabled(false);
      bStopWebcam.setEnabled(true);

      WebcamJTextArea.log("Gerät gestartet.");

      // Timer für Bildspeicherung und Verbindungsaufbau starten
      if (timer == null) {
         timer = new Timer();
      }
      timer.setTimerObjects(wPlayer, lastPic, pbNextPic);
      timer.start();
      WebcamJTextArea.log("Timer für Bildspeicherung und -übertragung gestartet.");

   }

});
```

Listing 19: Listener der Combobox cbDevices zur Auswahl und Start einer Webcam

Über die `CaptureDeviceInfo` wird der `MediaLocator` (Zugriffspfad) der Webcam besorgt. Der Manager setzt den Player, der hinfort die Webcam darstellt, in den Zustand „realized“ und übergibt ihn seiner Wrapper-Klasse `WebcamPlayer`. Mit dem Start des Players und des Abrufs der visuellen Darstellung des Players und deren Zuweisung an die grafische Komponente `webcamPic` wird der Bildstrom, den der Player liefert, auf der Oberfläche (`ClientPanel cp`) angezeigt. Die nächsten beiden Befehle bewirken, dass eine evtl. vorhandene andere Webcam erst ausgewählt werden kann, wenn der laufende Player zuvor gestoppt wurde. Anschließend wird der `Timer`, das zentrale Objekt der Applikation, mit den Objekten `WebcamPlayer` (die Webcam-Steuerung), `ImagePanel` (zeigt das aufgenommene Bild an) und `JProgressBar` (Anzeige der Dauer bis zum nächsten Bild) ausgestattet und gestartet.

Über den `Timer` wird der `ProgressTimer` gestartet und das erste Bild „gegrabbt“. Dazu wird ein zuvor erzeugtes Objekt der Klasse `Picture` benötigt, welches sich mit `wPlayer.getControl("javax.media.control.FrameGrabbingControl")` die Webcam-Steuerung besorgt und die Bilddaten über verschiedene Buffer einliest, anzeigt und in einem Verzeichnis temporär speichert. Damit der Fortschrittsbalken auch tatsächlich eine Veränderung anzeigen kann, wird er vom `Timer` mit der Methode `synchronize()` pro Bildspeicherung einmal synchronisiert. Dabei besorgt sich das Fortschrittsbalken-Objekt die aktuelle Zeit und berechnet den Zeitpunkt, wann nach Benutzervorgabe das nächste Foto geschossen werden muss. Dieser Zeitpunkt wird dann vom eigenen Thread alle 0,1 Sekunden mit der aktuellen Zeit in Relation gesetzt und der Fortschrittsbalken dementsprechend aktualisiert.

8.5 HTTPS-Upload der Bilddaten

Mit Speicherung des Bildes startet der `Timer` eine Verbindung (den Thread `Connection`). Je nachdem, wie Webcam-Client konfiguriert wurde, versucht der Thread nun die Bilddatei über eine bestehende oder eine noch zu öffnende Verbindung zu senden.
Bei einer bestehenden Verbindung wird wie folgt mit Hilfe der empfangenden DiLog-Servlet-Adresse getestet, ob die Verbindung noch besteht:

```
try {

   byte[] bbuffer = new byte[10];
   URL address = new URL("http://" + ClientSettings.getServerURL());
   InputStream is = address.openStream();
   int length = 0;

   while ((length = is.read(bbuffer, 0, 10)) != -1) {

      String s = new String(bbuffer, 0, length);

   }

} catch (Exception e) {

   WebcamJTextArea.log("Verbindung zum Server gescheitert.");
   return false;

}

WebcamJTextArea.log("Verbindung zum Server erfolgreich.");
return true;
```

Listing 20: Methode isConnectionAlive() zur Überprüfung der Verbindung

Falls dabei eine Exception geworfen wird, weil keine Daten gelesen werden können, wird versucht die Verbindung – wie im Falle einer gewollt nicht bestehenden Verbindung – mit `connect()` aufzubauen. Dazu wird die in Kapitel 8.2 schon erwähnte unter Windows ausführbare Datei `rasdial95.exe` benutzt.

Ein Objekt der Klasse `FileUpload` ist für die eigentliche Übertragung der Bilddaten verantwortlich. Da dies mit Passwortschutz und verschlüsselt geschehen soll,[57] muss erst noch SSL für selbstausgestellte Server-Zertifikate bereitgestellt werden. Dazu wird die Klasse `MySSLSocketFactory` benötigt. Dies ist deshalb der Fall, weil die Standard-SSL-Factory nicht mit eigenen selbstausgestellten Zertifikaten zurechtkommt. Zudem wird mit `EasyX509TrustManager` ein benötigter SSL-Trust-Manager zur Verfügung gestellt.
Die Übertragung der Daten übernimmt der Commons[58]-HttpClient. Das Framework HttpClient stellt Methoden für die clientseitige Arbeit mit dem HTTP-Protokoll bereit, die ohne Commons erst mühsam erstellt werden müssten. Hier kommt die Klasse `MultipartPostMethod` zum Einsatz. “Multipart” deshalb, weil der Request aus mehreren Teilen bestehen soll: Bilddatei, Kennung, Passwort und Datum. Der `HttpClient` sendet die Anfrage. Mit dem daraus resultierenden Status kann die Reaktion des Servers erfragt (`HttpStatus.getStatusText(status));`) und eine entsprechende Meldung im dafür vorgesehenen Ausgabefeld `WebcamJTextArea` ausgegeben werden.

```
MultipartPostMethod filePost =
   new MultipartPostMethod("https://" + ClientSettings.getServerURL());

try {

   File file = new File(ClientSettings.getPicsDir()
      + File.separator + ClientSettings.getPicname());

   filePost.addParameter(file.getName(), file);
   filePost.addParameter("identity", ClientSettings.getServerIdentity());
   filePost.addParameter("password", ClientSettings.getServerPassword());
   filePost.addParameter("date", new Date().getTime() + "");

   HttpClient client = new HttpClient();
   client.setConnectionTimeout(10000);
   int status = client.executeMethod(filePost);

   if (status == HttpStatus.SC_OK) {

      WebcamJTextArea.log("Bild übertragen. Server-Antwort: "
         + filePost.getResponseBodyAsString());

   } else {

      WebcamJTextArea.log("Bild konnte nicht übertragen werden. Server-Antwort: "
         + HttpStatus.getStatusText(status));

   }

} catch (Exception e) {

   WebcamJTextArea.log("Fehler bei der Bildübermittlung an den Server.");

} finally {
   filePost.releaseConnection();
}
```

Listing 21: Verwendung des Commons-HttpClient (MultipartPost)

[57] Ohne Kennung und Passwort könnten u.U. unautorisierte Clients mit etwas Glück oder Insider-Wissen unerwünschte Daten an den DiLog-Server senden.

[58] Commons ist ein Unterprojekt des Apache-Unterprojekts Jakarta. Commons hat die Bereitstellung von wiederverwendbaren Java-Komponenten zum Ziel.

9 Schlussbetrachtung

Ein wesentlicher Bestandteil der entstandenen Software DiLog ist das Framework Struts. Anfänglich war die Entwicklung mit Struts eher schleppend und wurde immer wieder durch Fehlermeldungen unterbrochen, für die nicht sofort eine Lösung ersichtlich war. Im Laufe des Projekts hat sich Struts jedoch als wahrer Segen herausgestellt, der die Entwicklung wirklich erleichterte. Als die grundlegenden Abläufe festgelegt und programmiert waren, konnte bei Implementierung von neuer Funktionalität immer wieder auf bereits vorhandene Schemen zurückgegriffen, ja sogar Source Code kopiert und teilweise mit nur leichten Modifizierungen übernommen werden. Die Tatsache, dass Struts nur über zwei Konfigurationsdateien (`struts-config.xml` und `tiles-config.xml`) zu steuern ist, trägt positiv zum Gesamtbild von Struts bei.

Insgesamt hat sich die Entwicklung von DiLog allerdings als wesentlich umfangreicher und komplizierter herausgestellt als geplant. Ein gut funktionierendes Software-System und dessen Entwicklung benötigt allerlei „Rankwerk“ wie Administration, Logging, Backup-Strategien, Konfiguration, Testläufe, Dokumentation und Datenübername aus einem Altsystem. Dies alles kostet Anstrengungen und vor allem Zeit. Zwar könnte man dies bei einem Prototypen alles klein halten, jedoch hatte ich den (für eine Diplomarbeit vielleicht übertriebenen) Ehrgeiz entwickelt, eine funktionierende Software zu erstellen. Wäre mir dies alles vor Beginn der Diplomarbeit klar gewesen, ich hätte mir wohl ein anderes Thema ausgesucht. Die Entwicklung eines Komplettsystems ist eben wesentlich umfangreicher als beispielsweise eine Machbarkeitsstudie oder ein abgegrenztes kleines Stück Software in einem großen Gesamtsystem.

Die Entwicklung eines Komplettsystems anstatt eines Prototyps kann auch noch aus einem weiteren Grund sehr zeitintensiv sein: Wenn man zu Perfektionismus neigt, kann man nicht mehr aufhören die Software voranzutreiben. Eine Schönheitskorrektur folgt der anderen, und man neigt dazu, sich hoffnungslos zu verzetteln. Zudem könnten noch so viele brauchbare und nützliche Funktionen implementiert werden…

Erschwerend kam hinzu, dass ich bei programmiertechnischen Fragen keinen direkten Ansprechpartner hatte, doch das war mir vor Beginn der Arbeiten schon klar. Ich möchte aber deutlich betonen, dass es mir Spaß gemacht hat, mich durchzubeißen und mich bald jeden Tag mit einem neuen unlösbar erscheinenden Problem konfrontiert zu sehen. Eine ständige und unerlässliche Hilfe war mir dabei das Internet mit all seinen Foren, Tutorials, Newsgroups, Mail-Archiven und Dokumentationen sowie der Kontakt über Chat und E-Mail zu gleichgesinnten Computer-Kennern und Freunden (siehe Dankesworte). Abschließend kann ich sagen, dass ich stolz und zufrieden auf die ca. 50.000 Code-Zeilen (ohne Leerzeilen) in ungefähr 400 Dateien blicke. Dabei handelt es sich zweifelsohne um mein bisher größtes und technisch vielseitigstes Software-Projekt, in dessen Verlauf ich jede Menge neue Erkenntnisse gewonnen habe. Trotz machen Durststrecken (auch privater Natur) möchte ich diese Zeit der Arbeit an DiLog nicht missen…

Literaturverzeichnis

[Alur 2002] ALUR, Deepak; CRUPI, John; MALKS, Dan
Core J2EE PATTERNS
Pearson Education Deutschland GmbH (Imprint: Markt+Technik Verlag), München 2002
(Core J2EE Patterns, Sun Microsystems Press, Prentice Hall PTR, 2001)

[Apache 2005] **Apache HTTP Server Project**
URL: http://httpd.apache.org (Stand: 29.01.2005)

[Balzert 2001] BALZERT, Heide
UML kompakt
Spektrum Akademischer Verlag GmbH, Heidelberg/Berlin 2001

[Cooper 2000] COOPER, James W.
Java design patterns : a tutorial
Thirdprinting, Addison-Wesley, USA 2000

[Edlich 2003] EDLICH, Stefan
Ant kurz & gut
1. Auflage 2002, korrigierter Nachdruck 2003, O'Reilly Verlag GmbH & Co. KG, Köln 2003

[Flanagan 1997] FLANAGAN, David
JavaScript – Das umfassende Referenzwerk
O'Reilly Verlag, Köln 1997
(JavaScript: The Definitive Guide, 2nd Edition, O'Reilly & Associates, Inc., 1997)

[Hall 2001] HALL, Marty
Core Servlets und JavaServer Pages
Pearson Education Deutschland GmbH (Imprint: Markt+Technik Verlag), München 2001
(Core Servlets and Java Server Pages, Sun Microsystems Press, Prentice Hall PTR, 2000)

[Horstmann 1999] HORSTMANN, Cay S.; CORNELL, Gary
Core JAVA 2, Band 1 – Grundlagen
Markt+Technik Buch- und Software-Verlag GmbH (Imprint: Prentice Hall), München 1999
(Core Java, Volume 1 – Fundamentals, Sun Microsystems, Inc, Prentice Hall, 1999)

[Java 2005] **Java Language Overview**
URL: http://java.sun.com/docs/overviews/java/java-overview-1.html (Stand: 29.01.2005)

[JK2 alt] **MOD_JK, Webserver to Tomcat Link**
URL: http://jakarta.apache.org/tomcat/connectors-doc-archive/jk2/index.html (Stand: 31.01.2005)

[JK2 neu] **The Apache Jakarta Tomcat Connector**
URL: http://jakarta.apache.org/tomcat/connectors-doc/index.html (Stand: 31.01.2005)

[Matzke 2003] MATZKE, Bernd
Ant - Das Java Build-Tool in der Praxis
Pearson Education Deutschland GmbH (Imprint: Addison-Wesley Verlag), München 2003

[OSI 2005] **Open Source Initiative (OSI)**
URL: http://www.opensource.org (Stand: 29.01.2005)

[Raymond 1997] RAYMOND, Eric
Die Kathedrale und der Basar
Linux-Magazin, Linux New Media AG, München 8/1997
URL: http://www.linux-magazin.de/Artikel/ausgabe/1997/08/Basar/basar.html
(The Cathedral & the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly & Associates)

[Reese 2003] REESE, George; YARGER, Randy; KING, Tim
MySQL Einsatz und Programmierung
2. Auflage, O'Reilly Verlag GmbH & Co. KG, Köln 2003
(Managing and Using MySQL, Second Edition, O'Reilly & Associates, Inc., 2002)

[Roßbach 2001] ROßBACH, Peter
Der neue Star im Jakarta-Projekt
In: Javamagazin 12/2001 S. 36
Software & Support Verlag GmbH, Frankfurt am Main 2001

[Seeboerger 2002] SEEBOERGER-WEICHSELBAUM, Michael
Das Einsteigerseminar Java/XML
verlag moderne industrie Buch AG & Co. KG, Bonn 2002

[Turner 2003] TURNER, James; BEDELL, Kevin
Struts – JSP-Applikationen mit Jakarta Struts, JBoss und Apache Axis
Pearson Education Deutschland GmbH (Imprint: Addison-Wesley Verlag), München 2003
(Struts Kick Start, Pearson Education, Inc., 2003)

[Ullenboom 2003] ULLENBOOM, Christian
Java ist auch eine Insel
2. Auflage, Galileo Press GmbH, Bonn 2003

[Ziegler 2002] ZIEGLER, Robert L.
Linux-Firewalls – Konzeption und Implementierung für Netzwerke und PCs
2. Auflage, Pearson Education Deutschland GmbH (Imprint: Markt+Technik Verlag), München 2002
(Linux Firewalls, 2nd Edition, Pearson Education, Inc., 2002)

Anhang

Servlet-Konfigurationsdatei web.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

   <display-name>DiLog - Digitales Logbuch</display-name>
   <description>Digitales Logbuch für den ASV</description>

   <!-- Diese Datei wird von Tomcat geladen -->
   <welcome-file-list>
      <welcome-file>/web/jsp/index.jsp</welcome-file>
   </welcome-file-list>

   <!-- Application Event Listener definieren -->
   <listener>
      <listener-class>magtime.helper.ShutDown</listener-class>
   </listener>

   <!-- Log4J-Init-Servlet -->
   <servlet>

      <servlet-name>log4j-init</servlet-name>
      <servlet-class>magtime.helper.Log4jInit</servlet-class>

      <init-param>
         <param-name>log4j-init-file</param-name>
         <param-value>conf/log4jconf.dilog.xml</param-value>
      </init-param>

      <load-on-startup>1</load-on-startup>

   </servlet>

   <!-- Config-Init-Servlet -->
   <servlet>

      <servlet-name>config-init</servlet-name>
      <servlet-class>magtime.helper.ConfigInit</servlet-class>

      <init-param>
         <param-name>config-init-file</param-name>
         <param-value>conf/applicationpath.properties</param-value>
      </init-param>

      <load-on-startup>1</load-on-startup>

   </servlet>

   <!-- Struts Action Servlet Configuration -->
   <servlet>

      <servlet-name>action</servlet-name>
      <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

      <init-param>
         <param-name>config</param-name>
         <param-value>/WEB-INF/struts-config.xml</param-value>
      </init-param>

      <init-param>
         <param-name>debug</param-name>
         <param-value>3</param-value>
      </init-param>

      <init-param>
         <param-name>detail</param-name>
         <param-value>3</param-value>
      </init-param>

      <load-on-startup>2</load-on-startup>
```

```
   </servlet>

   <!-- Struts Action Servlet Mapping -->
   <servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
   </servlet-mapping>

   <!-- Webcam Datei-Upload mit Commons HTTP-Client (geht nicht mit Struts) -->
   <servlet>
      <servlet-name>UploadServlet</servlet-name>
      <servlet-class>magtime.webcam.server.UploadServlet</servlet-class>
   </servlet>

   <!-- Webcam Servlet Mapping -->
   <servlet-mapping>
      <servlet-name>UploadServlet</servlet-name>
      <url-pattern>/servlet/UploadServlet</url-pattern>
   </servlet-mapping>

   <!-- Application Tag Library Descriptor -->
   <taglib>
      <taglib-uri>/WEB-INF/app.tld</taglib-uri>
      <taglib-location>/WEB-INF/app.tld</taglib-location>
   </taglib>

   <!-- Struts Tag Library Descriptors -->
   <taglib>
      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
   </taglib>

   <taglib>
      <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
   </taglib>

   <taglib>
      <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
   </taglib>

   <taglib>
      <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
   </taglib>

</web-app>
```

Tag Library Descriptor memberwrapper.tld:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
   "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>

   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>memberwrapper</shortname>

   <tag>

      <name>person</name>
      <tagclass>magtime.struts.taglibs.memberwrapper.PersonTag</tagclass>
      <bodycontent>empty</bodycontent>

      <attribute>
         <name>persId</name>
         <required>true</required>
         <rtexprvalue>true</rtexprvalue>
      </attribute>

      <attribute>
         <name>property</name>
         <required>true</required>
      </attribute>

      <attribute>
         <name>unload</name>
```

```
         <required>false</required>
      </attribute>

   </tag>

   <tag>

      <name>member</name>
      <tagclass>magtime.struts.taglibs.memberwrapper.MemberTag</tagclass>
      <bodycontent>empty</bodycontent>

      <attribute>
         <name>persId</name>
         <required>true</required>
         <rtexprvalue>true</rtexprvalue>
      </attribute>

      <attribute>
         <name>property</name>
         <required>true</required>
      </attribute>

      <attribute>
         <name>unload</name>
         <required>false</required>
      </attribute>

      <attribute>
         <name>boatId</name>
         <rtexprvalue>true</rtexprvalue>
         <required>false</required>
      </attribute>

   </tag>

</taglib>
```

Ant-Build-Datei buildDiLog.xml (Kennung und Passwörter wurden durch abc bzw. xyz ersetzt):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<project name="buildDiLog" default="main" basedir="E:/server/tomcat5/webapps/dilog">

<!-- Hinweis: Das Verzeichnis webapps/dilog muss manuell angelegt werden -->

<!-- Dateipfade aus externer Properties-Datei laden -->
<property file="D:/da_asv_hh/eclipse/ant/dilog.properties" prefix="prop"/>

<!-- Properties für die Tomcat Manager Applikation -->
<property name="url" value="http://localhost:8080/manager/html"/>
<property name="username" value="abc"/>
<property name="password" value="xyz"/>

<!-- Ant Tasks für Tomcat Manager Applikation-->
<taskdef name="deploy" classname="org.apache.catalina.ant.DeployTask" />
<taskdef name="list" classname="org.apache.catalina.ant.ListTask"/>
<taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask"/>
<taskdef name="resources" classname="org.apache.catalina.ant.ResourcesTask"/>
<taskdef name="roles" classname="org.apache.catalina.ant.RolesTask"/>
<taskdef name="start" classname="org.apache.catalina.ant.StartTask"/>
<taskdef name="stop" classname="org.apache.catalina.ant.StopTask"/>
<taskdef name="undeploy" classname="org.apache.catalina.ant.UndeployTask"/>

<!-- Classpath zum Kompilieren -->
<path id="cp">
   <pathelement path="${classpath}" />
   <pathelement path="WEB-INF/lib/log4j-1.2.8.jar" />
   <pathelement path="WEB-INF/lib/junit.jar" />
   <pathelement path="WEB-INF/lib/servlet-api.jar" />
   <pathelement path="WEB-INF/lib/struts-1.1b2.jar" />
   <pathelement path="WEB-INF/lib/commons-beanutils.jar" />
   <pathelement path="WEB-INF/lib/jsp-api.jar" />
   <pathelement path="WEB-INF/lib/mail.jar" />
   <pathelement path="WEB-INF/lib/activation.jar" />

   <!-- nur zum Kompilieren von nicht für den Server benötigtem Code -->
   <pathelement path="WEB-INF/lib/commons-httpclient-2.0.jar" />
   <pathelement path="WEB-INF/lib/commons-fileupload-newversion.jar" />
   <pathelement path="WEB-INF/lib/LightJBCL3.jar" />
```

```
   <pathelement path="WEB-INF/lib/jmf.jar" />
   <pathelement path="WEB-INF/lib/iText.jar" />

</path>

<!-- normaler Kompilierungsvorgang (alles, inkl. vorheriges  Löschen aller vorhandenen
     Dateien) -->
<target name="main" depends="makedirs, clean, makedirs, java, clientjars">
</target>


<!-- Für Java-Files und bei Änderungen in Ressourcen- oder Konfigurationsdateien Kopiert nur
     Dateien, muss also nach einem einmaligen kompletten Build ausgeführt werden -->
<target name="java" depends="copy, compile, reload">
</target>


<!-- Für Java-Files und bei Änderungen in Ressourcen- oder Konfigurationsdateien Kopiert nur
     Dateien, muss also nach einem einmaligen kompletten Build ausgeführt werden -->
<target name="javalight" depends="copy, compileJava, reload">
</target>


<!-- Für JSPs ist kein Kompilieren notwendig, da sie vom Servlet-Container sowieso nochmals
     kompiliert werden. Kopiert nur Dateien, muss also nach einem kompletten Build ausgeführt
     werden. Ohne Syntaxprüfung! -->
<target name="jsp" depends="copy">
</target>


<!-- Kopiert nur Dateien, muss also nach einem einmaligen kompletten Build ausgeführt werden.
     Mit Syntaxprüfung! -->
<target name="jspsyntaxcheck" depends="copy, compileJSP">
</target>


<!-- Erstellt eine WAR-Datei für den Servlet-Container. Kopiert auch die MySQL-Datenbank. -->
<target name="war" depends="makedirs, clean, makedirs, copy, compile, makewar, copydb">
</target>


<!-- Java-Dateien + JSP-Dateien. Für JSPs mit Syntaxprüfung werden alle JSPs in java-Dateien
     umgewandelt. Wenn dabei ein Fehler auftritt stoppt der Build. Vorher werden class-Dateien
     erstellt, da die JSPs abhängig sind von diesen (eigene Taglibs!). Mit Syntaxprüfung! -->
<target name="compile" depends="compileJava, compileApplet, signjar, compileJSP">
</target>


<!-- Kopiert abhängig von der Zielplatform die entsprechenden Konfigurationsdateien -->
<target name="makewar" depends="copyconf_entw, copyconf_prod">

   <delete failonerror="false">
      <fileset dir="${prop.warexportpath}" includes="dilog.war" />
   </delete>

   <war destfile="${prop.warexportpath}/dilog.war"
      webxml="${prop.sourcepath}/WEB-INF/web.xml">
      <fileset dir=".">
         <include name="**/*" />
         <exclude name="**/*.bak" />
         <exclude name="**/*.java" />
         <exclude name="WEB-INF/web.xml" />
      </fileset>
   </war>

   <!-- Win-Konfigdateien wieder herstellen -->
   <copy todir="conf" overwrite="true">
      <fileset dir="${prop.sourcepath}/conf" includes="*.*" />
      <fileset dir="${prop.eclipsepath}/conf" includes="*.dilog.xml" />
   </copy>

</target>


<!-- Linux-Konfigdateien für Entwicklungsserver kopieren. -->
<target name="copyconf_entw" if="prop.war_entw">

   <copy todir="conf" overwrite="true">
      <fileset dir="${prop.sourcepath}/conf/linux_entw" includes="*.*" />
      <fileset dir="${prop.eclipsepath}/conf/linux_entw" includes="*.dilog.xml" />
   </copy>
```

```
</target>


<!-- Linux-Konfigdateien für Produktivserver kopieren. -->
<target name="copyconf_prod" if="prop.war_prod">

   <copy todir="conf" overwrite="true">
      <fileset dir="${prop.sourcepath}/conf/linux_prod" includes="*.*" />
      <fileset dir="${prop.eclipsepath}/conf/linux_prod" includes="*.dilog.xml" />
   </copy>

</target>


<!-- Kopiert die MySQL-Datenbank und ändert die Kleinschreibung des ersten Buchstabens der
     Tabellennamen in Großschreibung.
     Achtung: Nur Tabellen vom Typ MyISAM sind plattformunabhängig, InnoDB-Tabellen
     funktionieren nicht auf anderen Systemen wenn kopiert.
     Achtung: Der MySQL-Dump funktioniert nicht mit Inno-DB, da die Wiederherstellung an den
     Fremdschlüsseln scheitert (falsche Reihenfolge der CREATE- und INSERT-Anweisungen bei der
     Generierung des SQL-Skipts durch mysqldump). Dieses Target wird nicht mehr gebraucht,
     stattdessen StartDatatransformation.java benutzen und jeweiligen Ziel-Host in der XML-
     Konfigurationsdatei angeben!-->
<target name="copydb">

   <!-- Version MyISAM -->

   <copy todir="${prop.warexportpath}/MyISAM" overwrite="true">
      <fileset dir="${prop.dbpath}/Asv" includes="*" />
   </copy>

   <java classname="magtime.helper.ConvertTablenames" classpath="D:/da_asv_hh/eclipse/bin">
      <arg value="${prop.warexportpath}/Asv" />
   </java>

   <!-- Version InnoDB -->

   <exec executable="mysqldump">
      <arg line="--opt -abc -xyz -rD:/da_asv_hh/export/InnoDB/asvdbdump Asv" />
   </exec>

</target>


<!-- Java-Dateien kompilieren -->
<target name="compileJava">

   <delete includeemptydirs="true">
      <fileset dir="WEB-INF/backup" includes="**/*" />
   </delete>
   <move todir="WEB-INF/backup">
      <fileset dir="WEB-INF/src" includes="**/*" />
      <mapper type="glob" from="*.java" to="*.bak" />
   </move>
   <copy todir="WEB-INF/src">
      <fileset dir="${prop.eclipsepath}/src" includes="**/*" />
   </copy>

   <!-- Kompilieren aller Klassen außer Applet-Klassen -->
   <javac srcdir="WEB-INF/src" destdir="WEB-INF/classes">
      <classpath refid="cp" />
      <exclude name="magtime/chat/client/*.java" />
   </javac>

</target>


<!-- Java-Applet-Dateien kompilieren und Applet-JAR erstellen -->
<target name="compileApplet">

   <!-- Kompilieren der Applet-Klassen mit JDK 1.1.x -->
   <javac srcdir="WEB-INF/src" destdir="WEB-INF/classes" target="1.1">
      <classpath refid="cp" />
      <include name="magtime/chat/client/*.java" />
   </javac>

   <jar destfile="web/applet/chatapplet.jar">
      <fileset dir="WEB-INF/classes">
         <include name="magtime/chat/client/*.class" />
      </fileset>
      <fileset dir="${prop.eclipsepath}/conf/chatapplet" includes="*.*" />
   </jar>
```

```
</target>


<!-- Erstellt ein Schlüselpaar und signiert damit das Chat-Applet -->
<target name="signjar">

   <!-- Schlüsselpaar löschen, da genkey abbricht, wenn Alias schon vorhanden -->
   <exec executable="keytool">
      <arg line="-delete -alias dilog -storepass abc" />
   </exec>

   <genkey alias="dilog" storepass="abc" keypass="abc" validity="10000" keyalg="rsa">
      <dname>
         <param name="CN" value="Steffen Hartmann"/>
         <param name="OU" value="DiLog"/>
          <param name="O"  value="MSP Hartmann"/>
          <param name="C"  value="DE"/>
      </dname>
   </genkey>

   <exec executable="keytool">
      <arg line="-selfcert -alias dilog -storepass abc" />
   </exec>

   <!-- unsigniertes Applet kopieren und umbenennen -->
   <copy file="web/applet/chatapplet.jar" tofile="web/applet/chatapplet_signed.jar" />

   <signjar alias="dilog" storepass="abc" keypass="abc">
      <fileset dir="web/applet" includes="chatapplet_signed.jar" />
   </signjar>

</target>


<!-- Client-JARs (IETool + Webcam-Client) erstellen -->
<target name="clientjars" depends="copy, compileJava">

   <mkdir dir="${prop.warexportpath}/ietool" />
   <mkdir dir="${prop.warexportpath}/webcamclient" />

   <delete failonerror="false">
      <fileset dir="${prop.warexportpath}/ietool" includes="*.jar" />
      <fileset dir="${prop.warexportpath}/webcamclient" includes="*.jar" />
   </delete>

   <copy todir="${prop.warexportpath}/ietool">
      <fileset dir="${prop.sourcepath}/WEB-INF/lib" includes="log4j-1.2.8.jar,
         commons-logging.jar, LightJBCL3.jar, mysql-connector-java-3.0.9-stable-bin.jar" />
   </copy>
   <copy todir="${prop.warexportpath}/webcamclient">
      <fileset dir="${prop.sourcepath}/WEB-INF/lib" includes="log4j-1.2.8.jar,
         commons-logging.jar, LightJBCL3.jar, jmf.jar commons-httpclient-2.0.jar" />
   </copy>

   <jar destfile="${prop.warexportpath}/ietool/ietool.jar"
      manifest="${prop.eclipsepath}/conf/ietool/manifest.mf">
      <fileset dir="WEB-INF/classes">
         <include name="magtime/ietool/*.class" />
         <include name="magtime/config/**/*.class" />
         <include name="magtime/db/**/*.class" />
         <include name="magtime/objectlevel/datacontainer/DataContainer.class" />
      </fileset>
   </jar>
   <jar destfile="${prop.warexportpath}/webcamclient/webcamclient.jar"
      manifest="${prop.eclipsepath}/conf/webcamclient/manifest.mf">
      <fileset dir="WEB-INF/classes">
         <include name="magtime/webcam/client/*.class" />
      </fileset>
   </jar>

</target>


<!-- JSP-Dateien kompilieren (Syntaxprüfung) -->
<target name="compileJSP">

   <mkdir dir="jsptemp" />

   <taskdef classname="org.apache.jasper.JspC" name="jasper2">
      <classpath>
         <fileset dir="${prop.tomcatpath}/bin">
```

```
            <include name="*.jar"/>
         </fileset>
         <fileset dir="${prop.tomcatpath}/server/lib">
            <include name="*.jar"/>
         </fileset>
         <fileset dir="${prop.tomcatpath}/common/lib">
            <include name="*.jar"/>
         </fileset>
      </classpath>
   </taskdef>

   <jasper2 uriroot="${prop.applicationpath}" outputDir="${prop.applicationpath}/jsptemp" />

   <delete includeemptydirs="true" failonerror="false">
      <fileset dir="jsptemp" includes="**/*" />
      <fileset dir="" includes="jsptemp" />
   </delete>

</target>


<target name="copy">

   <copy todir="web">
      <fileset dir="${prop.sourcepath}/web" includes="**/*" />
   </copy>
   <copy todir="WEB-INF">
      <fileset dir="${prop.sourcepath}/WEB-INF" includes="*.*" />
   </copy>
   <copy todir="WEB-INF/lib">
      <fileset dir="${prop.sourcepath}/WEB-INF/lib" includes="*" />
   </copy>
   <copy todir="conf">
      <fileset dir="${prop.sourcepath}/conf" includes="*.*" />
      <fileset dir="${prop.eclipsepath}/conf" includes="*.dilog.xml" />
   </copy>
   <copy todir="conf/dtd">
      <fileset dir="${prop.eclipsepath}/conf/dtd" includes="*.dilog.dtd" />
   </copy>
   <copy todir="WEB-INF/classes">
      <fileset dir="${prop.sourcepath}/resources" includes="*.properties" />
   </copy>

</target>


<target name="clean">

   <delete includeemptydirs="true" failonerror="false">
      <fileset dir="WEB-INF/classes" includes="**/*" />
      <fileset dir="WEB-INF/lib" includes="**/*" />
      <fileset dir="WEB-INF" includes="*.*" />
      <fileset dir="web" includes="**/*" />
      <fileset dir="conf" includes="**/*" />
   </delete>

</target>


<target name="makedirs">

   <mkdir dir="conf" />
   <mkdir dir="conf/dtd" />
   <mkdir dir="logs" />
   <mkdir dir="web" />
   <mkdir dir="temp" />
   <mkdir dir="webcamtemp" />
   <mkdir dir="webcampics" />
   <mkdir dir="WEB-INF" />
   <mkdir dir="WEB-INF/classes" />
   <mkdir dir="WEB-INF/lib" />
   <mkdir dir="WEB-INF/src" />
   <mkdir dir="WEB-INF/backup" />

</target>


<target name="javadoc">

   <javadoc destdir="${prop.eclipsepath}/javadoc" access="private" use="true" notree="false"
      nonavbar="false" noindex="false" splitindex="true" author="true" version="true"
      nodeprecatedlist="false" nodeprecated="false"
```

```
      packagenames="magtime.reservation,magtime.testing,magtime.db.importmysqldb,magtime.db,ma
      gtime.struts.guest,magtime.objectlevel,magtime.struts,magtime.webcam.client,magtime.pdf,
      magtime.db.objectlevel,magtime.struts.user,magtime.db.exportmysqldb,magtime.config,magti
      me.struts.helper,magtime.struts.taglibs.memberwrapper,magtime.db.datatransformation,magt
      ime.helper,magtime.struts.login,magtime.struts.admin,magtime.objectlevel.datacontainer,m
      agtime.trial,magtime.chat.server,magtime.webcam.server,magtime.chat.client,magtime.confi
      g.xml,magtime.struts.beans,magtime.db.exportaccessdb,magtime.start,magtime.reservation.r
      ules,magtime.testing.start,magtime.struts.foreman,magtime.mail,magtime.ietool,magtime.te
      sting.manual,magtime.db.importaccessdb,magtime.testing.junit"
      sourcepath="${prop.eclipsepath}/src"
      classpath="bin;lib\log4j-1.2.8.jar;lib\mysql-connector-java-3.0.9-stable-
      bin.jar;E:\winXP\eclipse\plugins\org.junit_3.8.1\junit.jar;E:\server\tomcat5\common\lib\
      servlet-api.jar;lib\struts-1.1b2.jar;lib\commons-beanutils.jar;D:\da_asv_hh\dilog\WEB-
      INF\lib\jsp-api.jar;D:\da_asv_hh\dilog\WEB-
      INF\lib\activation.jar;D:\da_asv_hh\dilog\WEB-INF\lib\mail.jar;lib\commons-httpclient-
      2.0.jar;lib\commons-logging-api.jar;lib\commons-logging.jar;lib\commons-fileupload-
      newversion.jar;lib\LightJBCL3.jar;lib\jmf.jar;lib\iText.jar" doctitle="JavaDoc für DiLog
      (Digitales Logbuch)" additionalparam="" />

</target>


<!-- Ant gibt BUILD FAILED aus, reload hat aber trotzdem funktioniert -->
<target name="reload">

   <reload url="${url}" username="${username}"
      password="${password}" path="/dilog" />

</target>


</project>
```

Konfigurationsdatei dilog.properties für Ant-Build-Datei:

```
sourcepath=D:/da_asv_hh/dilog
eclipsepath=D:/da_asv_hh/eclipse
warexportpath=D:/da_asv_hh/export
tomcatpath=E:/server/tomcat5
applicationpath=E:/server/tomcat5/webapps/dilog
dbpath=D:/da_asv_hh/diplomarbeitsmaterial/db-design/mysqldata

# Es darf nur einer der drei Werte gesetzt sein!
#  war_prod = Linux-Produktivsystem, war_entw = Linux-Entwicklungssystem, war_win = Windows-
Entwicklungssystem
war_prod=true
```